Modeling the adversary to evaluate password strength with limited samples

Thesis Proposal

Author: Saranga Komanduri

Committee: Dr. Lorrie Cranor (Chair)
Dr. Lujo Bauer
Dr. Nicolas Christin
Dr. Paul Van Oorschot (Carleton University)

For the Ph.D. Program in Computation, Organizations and Society
Institute for Software Research
School of Computer Science
Carnegie Mellon University

October 2013
# Table of Contents

Abstract iii  
1 Motivation 1  
2 Approach 2  
3 Background 3  
4 Summary of Contributions 8  
5 The Guess Calculator Framework 10  
6 Measuring Password Guessability for an Entire University 20  
7 Remaining Work 25  
8 Overview of Related Work 34  
9 Chapter Outline 43  
10 Timeline 45  
Bibliography 46
Abstract

In an effort to improve security by preventing users from picking weak passwords, system administrators set policies, sets of requirements that passwords must meet. Guidelines for such policies have been published by various groups but this guidance has not been empirically verified. In fact, our research group and others have discovered it to be inaccurate [79, 87].

The goal of this thesis is to provide an improved metric for evaluating the security of password policies, compared to previous machine-learning approaches. We make five major contributions to passwords research. First, we develop a guess calculator framework that learns a model of adversary guessing from a training set of prior data mixed with samples, and use this framework to evaluate passwords. Second, since the adversary’s best strategy is to guess passwords based on the likelihood with which users create them, we conduct a thorough analysis of user-created passwords and the password-creation process. Third, we use the guess calculator framework to evaluate the guessability of passwords for an entire university and compare this to other sources of passwords, namely subsets of leaked datasets and passwords collected in research studies. This allows us to measure the usefulness of these sources. Fourth, we make several improvements to the guessing model to increase the power of the learning algorithm and improve guessing efficiency. Fifth, we provide guidance on using the framework for policy evaluation.
1 Motivation

Passwords might be the most popular form of computer-human authentication currently in use. Though a wide array of alternative forms of authentication have been proposed: biometrics [2], graphical password schemes [3], multi-factor authentication [68], schemes that use a smartphone [58], keystroke dynamics [65], etc., simple text passwords are used extensively on websites and in the enterprise. This would be an acceptable state of affairs if passwords were an ideal solution. Unfortunately, the decreasing cost of computation requires us to have increasingly secure password systems. There have been several high-profile password breaches in recent years in which hashed password files were stolen [22, 47–49, 55, 64, 71, 75, 80, 83]. Though the passwords are hashed, they are still vulnerable to a “guessing attack” in which the adversary generates guesses and can verify these guesses using the file [63]. Organizations seem to respond to the threat of guessing attacks by making password policies more complex, but many researchers have identified a so-called “password problem” [33]: more complex passwords can be more difficult to remember. While cryptography research is discovering better ways to resist online attacks [1] and users could use password managers to alleviate some of the password problem [35], the need for users to create and use text passwords does not appear to be going away anytime soon [7, 39].

The difficulties with passwords are also well known outside of the research community. In fact, they have received plenty of coverage in the popular press since the public began to use passwords with online services, and this coverage has steadily increased with time. To investigate the popularity of passwords as a topic, I queried the Proquest reference database for articles in newspapers and magazines that contain both of the words “password” and “security.” The result is shown in Figure 1. The blue line shows a trend in the number of articles over time that steadily increases from 1990 onwards, mirroring the growth of online services.

Yet even with all this attention, the state of the art in setting password policies has not advanced. In 2006, the National Institute of Standards and Technology (NIST) published guidelines for setting password policies [14]. This guidance was based on many assumptions about user behavior that our research group subsequently found to be inadequate [79] which led to inaccurate estimates of password strength [46, 52]. The guidelines were revised in 2011 to remove many of
Without an adequate metric for password policies, organizations will continue to make uninformed decisions. Unfortunately, accurately characterizing password distributions requires more samples than are feasible to collect. After collecting statistics on 69 million Yahoo! passwords, Bonneau found that 42.5% were unique in the dataset; even with millions of samples, accurate probabilities cannot be assigned to almost half of the observed passwords. Further, when Bonneau fit a parametric model to the frequency distribution of the data, this model was significantly different from the observed distribution [6]. Taken together, these facts suggest that the current state of the art in policy metrics suffers from a lack of data and might not generalize to new policies. When evaluating new or proposed password policies, it is usually not feasible to collect statistics on millions of passwords, so there is a need for metrics that are applicable to relatively small samples. Few metrics of this type have been published previously, and many researchers find those that have been published unsatisfying [5, 87].

2 Approach

Figure 1: Number of newspaper and magazine articles per month that mention passwords. The blue trend line is a loess (“local regression”) line which fits a curve to data in a local, nonparametric manner. The gray shading around the line represents a 95% confidence interval for the trend.

the earlier recommendations, and new guidance was not inserted in its place [13]. The NIST guidelines had broad impact, however, and were cited as an authority in the password policy of several organizations, Carnegie Mellon included [79].
The goal of this thesis is to provide an improved metric for evaluating proposed password policies with relatively small samples compared to previous machine-learning approaches. We accomplish this by modeling guessing attacks using a formal grammar under a limited-knowledge, offline-attack threat model. One policy is considered stronger than another if passwords from that policy take longer to guess. This approach learns from the content of passwords as opposed to other metrics which focus just on their frequency distribution [6]. This increases the amount of data available to the model, even with small samples. We believe this approach is realistic; it is reasonable to assume that adversaries also lack knowledge of the true distribution of passwords, given the amount of data that would be required to gain such knowledge, but they still attempt to guess passwords as accurately as possible given the data available.

It is known that white-hat hackers incorporate experience and intuition into their attacks, manually identifying patterns in passwords in order to crack them more efficiently [34]. This thesis, in contrast to a manual approach, uses machine learning to learn password patterns automatically using natural-language processing techniques. By applying machine-learning algorithms to a mix of prior data and samples, we produce a generative model of adversary guesses to compare against target passwords. This provides a framework for studying passwords in a more principled, reproducible way.

3 Background

In this section, I define terms and introduce concepts that will be used often in this document. Related work will be cited where needed, but a more thorough review of related work is in Section 8.

3.1 Policies

Password policies impose hard constraints on users during password creation. We call these requirements. For example, a policy might require that passwords meet a minimum length, have a particular number of symbols, or pass a dictionary check.

The password-creation environment might impose soft constraints on users as well. When creating or entering a password on a mobile device, for example, there is a usability cost associated with using special characters. It is possible to use
them, but some users might prefer not to. The impact of such soft constraints is unknown and could lead to passwords that are less secure than expected.

For simplicity, we call the complete set of constraints that passwords are created under a policy even if this includes things like user constraints that are not set by the system administrator. When adversaries try to guess a set of passwords, we say that they are making guesses against a target policy. Soft constraints might also affect the distribution of passwords, so an intelligent adversary should be expected to take advantage of them when attacking passwords.

It is also helpful to classify policies as simple or complex. Previous work focused on datasets that were created under very simple policies: policies with a 6–8 character minimum length requirement and possibly one other requirement such as a digit, symbol, or uppercase letter. Leaked password sets have given us insight into simple policies by providing us with millions of passwords, but the same cannot be said of complex policies, where this amount of data is unavailable. We are also concerned with modeling passwords created under soft constraints, such as passwords created under mobile devices. While the requirements for such a policy might be simple, we lack data about how usability constraints affect passwords. Measuring the strength of passwords that users create under complex policies is an open and important problem, as is understanding how soft constraints might affect passwords. The methodology of this thesis is applicable to both types of problems.

### 3.2 Tokenizing

It is well known that many passwords contain semantic elements including names or dictionary words [15, 62, 76, 92]. Tokenizing, as used in this thesis, is the process of splitting a password into these semantic elements or tokens. The general approach is to develop a language model from existing data, where words are separated by spaces, and apply this model to find likely splits in unseparated text. This approach is applied in this thesis, largely influenced by the tools made available by Wang et al. [84] who base their models on data from the Bing web crawler. For this thesis, I use data from the publicly released Google Web Corpus [9].
3.3 Using formal languages to model passwords

For a given password and guessing algorithm, “guessability” measures the number of guesses needed by the algorithm to guess that password. In this thesis, we model passwords with a formal, probabilistic grammar. Our guessing algorithm uses this grammar to generate guesses in probability order, starting with the most probable string that can be generated by the grammar.

As explained earlier, passwords datasets are dominated by “singletons,” passwords that only appear once in the data set. In samples of 1,000 passwords, it is common for the incidence of singletons to be greater than 95% of the sample. Therefore, a useful guessing algorithm must be able to produce guesses that do not appear in training data, otherwise singletons will never be guessed. Traditional password crackers achieve this with “mangling rules.” These are rules that can be applied to words from an input dictionary to generate new guesses outside of the dictionary [60].

“Guessing efficiency” refers to the rate of success of a guessing attack per guess. Mangling rules can be used to produce a large number of guesses, but they might have a very low rate of success. Both Narayanan et al. and Weir et al. proposed probabilistic models of guessing [67,88]. In these models, many guesses can be assigned probability values—including guesses that do not appear in the training data. When guesses are made in probability order, these models can provide a significant advantage in guessing efficiency over traditional password crackers [88].

3.3.1 Probabilistic context-free grammars

The probabilistic context-free grammars (PCFGs) used by Weir et al. are a key component of this thesis. Following is a review of PCFGs, a discussion of the form of Weir’s PCFG, and the introduction of two terms that are unique to this thesis: structures and patterns.

A probabilistic context-free grammar can be defined as a 5-tuple of finite sets $\langle \Sigma, \mathcal{N}, S, R, \Theta \rangle$, where $\Sigma$ is a set of terminals, $\mathcal{N}$ is a set of nonterminals, $S$ is a special element of $\mathcal{N}$ called the start symbol, $R$ is a set of rules having the form $A \rightarrow \xi$ with $A \in \mathcal{N}$ and $\xi \in (\Sigma \cup \mathcal{N})^*$, and $\Theta = \{\theta_{A \rightarrow \xi} : A \rightarrow \xi \in R\}$ is a one-to-one mapping of probabilities to rules such that

$$\forall A \in \mathcal{N}, \sum_{(A \rightarrow \xi) \in R} \theta_{A \rightarrow \xi} = 1$$
See [19] for more background on probabilistic context-free grammars.

**Weir’s PCFG**  With the above definition in mind, we find that the PCFG used by Weir et al. is relatively simple. Define $Q = (\mathcal{N} \setminus S)$ as the set of nonterminals minus the start symbol. All production rules in Weir’s PCFG are in one of two forms:

\[
\begin{align*}
S & \rightarrow Q \\
Q & \rightarrow T 
\end{align*}
\]

The grammar is non-recursive; aside from the start symbol, no nonterminal produces nonterminals. Additionally, Weir’s PCFGs are defined so that all terminals that replace a particular nonterminal are the same length and each character in the terminal is of the same class. In other words, we can write $Q = \{L_i, D_i, S_i\}, \forall i \in [1, M]$ for some maximum length $M$, where $L$ represents alphabetic strings, $D$ digits, $S$ special characters, and $i$ the string length of replacement terminals.

**Producing guesses**  We can derive a PCFG from a training corpus. For example, the set of passwords \{password!, password!, baseball!, baseball123!\} can be parsed to produce the simple PCFG shown in Figure 2. The mapping from training corpus to PCFG is not one-to-one; there are other possible training corpora that could produce this PCFG.

Beginning with the start symbol, one can repeatedly apply the rules of a grammar until a string with only terminals is produced. The set of all strings that can be produced in this way is called a “language” [19], and each string in the language has an associated probability equal to the product of all rules’ probabilities used in its production. To produce guesses using a PCFG, we want to produce strings in the language in descending order of probability. Weir et al. provide an algorithm for doing this using a priority-queue that stores potential “next” guesses [86].

Weir et al. use the term *structure* to refer to the string of nonterminals on the right-hand side of Rule (3.3.1). Our example grammar contains two structures that represent strings of length 12 and 9, $L_8D_3S_1$ and $L_8S_1$. Structures are specific to Weir’s PCFG and are an essential, high-level component of how we model passwords. Only guesses that match a previously learned structure can be produced by Weir’s PCFG, but a single structure can produce thousands or millions of guesses.
\[\Sigma: \{\text{password, baseball, 123, !}\} \quad \text{(terminals)}\]

\[\mathcal{N}: \{L_8, D_3, S_1\} \quad \text{(nonterminals)}\]

\[\mathcal{R}: S \rightarrow L_8D_3S_1 \quad \text{(structures)}\]

\[S \rightarrow L_8S_1\]

\[L_8 \rightarrow \text{password}\] \quad \text{(terminal productions)}

\[L_8 \rightarrow \text{baseball}\]

\[D_3 \rightarrow 123\]

\[S_1 \rightarrow !\]

\[\Theta: \theta_{S \rightarrow L_8D_3S_1} = 0.25 \quad \text{(probabilities)}\]

\[\theta_{S \rightarrow L_8S_1} = 0.75\]

\[\theta_{L_8 \rightarrow \text{password}} = 0.5\]

\[\theta_{L_8 \rightarrow \text{baseball}} = 0.5\]

\[\theta_{D_3 \rightarrow 123} = 1.0\]

\[\theta_{S_1 \rightarrow !} = 1.0\]

**Figure 2**: Example PCFG of the form used by Weir et al. [88] for modeling passwords

The guesses produced by our simple PCFG from Figure 2 are shown in Figure 3. This PCFG is only capable of producing four guesses. Note that the first two and last two pairs of guesses share the same probability and the same structure. We define a **pattern** as a representation of a group of guesses that share the same structure **and** the same probability. Representing groups of guesses as patterns allows us to store guesses in a more compact form. This fact is used in generating the “lookup table” that is an essential component of this thesis (discussed in Section 5.2).

<table>
<thead>
<tr>
<th>Guess #</th>
<th>Guess</th>
<th>Probability</th>
<th>Probability derivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>password!</td>
<td>0.375</td>
<td>[\theta_{S \rightarrow L_8S_1} \cdot \theta_{L_8 \rightarrow \text{password}} \cdot \theta_{S_1 \rightarrow !}]</td>
</tr>
<tr>
<td>2</td>
<td>baseball!</td>
<td>0.375</td>
<td>[\theta_{S \rightarrow L_8S_1} \cdot \theta_{L_8 \rightarrow \text{baseball}} \cdot \theta_{S_1 \rightarrow !}]</td>
</tr>
<tr>
<td>3</td>
<td>password123!</td>
<td>0.125</td>
<td>[\theta_{S \rightarrow L_8D_3S_1} \cdot \theta_{L_8 \rightarrow \text{password}} \cdot \theta_{D_3 \rightarrow 123} \cdot \theta_{S_1 \rightarrow !}]</td>
</tr>
<tr>
<td>4</td>
<td>baseball123!</td>
<td>0.125</td>
<td>[\theta_{S \rightarrow L_8D_3S_1} \cdot \theta_{L_8 \rightarrow \text{baseball}} \cdot \theta_{D_3 \rightarrow 123} \cdot \theta_{S_1 \rightarrow !}]</td>
</tr>
</tbody>
</table>

**Figure 3**: Guesses produced by the example PCFG of Figure 2.
4 Summary of Contributions

The new technical contributions of this thesis are:

**The guess calculator framework** Using Weir’s probabilistic context-free grammar (PCFG) [88] to determine guess numbers for passwords, implemented in a manner that operates much faster than explicitly enumerating individual guesses.

**Deep analysis of password creation** An in-depth analysis of semantic elements in passwords for various datasets.

**Guessability for an entire university** A comparison of passwords from Carnegie Mellon’s single-sign-on system to passwords collected from Mechanical Turk or found in leaked password sets.

**Improvements to the guessing model** Improvements to Weir’s PCFG to better model an intelligent adversary: string frequency information, tokenizing strings, and using a more powerful grammar.

**Methodological guidance** Guidance for conducting experiments with the guess calculator framework.

The various components of this thesis are presented in Figure 4 and include required data sources, the technical contributions, and various products. Items that are already complete are shown in dark green. Remaining work is unshaded. Each of the contributions listed above will be treated as separate chapters in the thesis.

The first technical contribution, developing the guess calculator framework, is already done and will be presented in Section 5. The “deep analysis” is nearly finished and suggests that improvements to Weir’s PCFG could improve the efficiency of our guessing model. As can be seen, measuring guessability at CMU uses the guess calculator framework but not improvements to the guessing model. This is because the availability of plaintext passwords was time-sensitive; the passwords were only available under an older identity-management system that has already been replaced. This part of the thesis is also complete and presented in Section 6. After improvements to the guessing model are complete, we will reevaluate the policies that have already been collected, in the manner of the results that will be presented in Section 5.3. Finally, we will conduct additional
experiments similar to those described in Section 5.3.3 and compile what was learned into methodological guidance.

![Diagram of contributions]

Figure 4: The contributions of this thesis and their relationship to each other and previous work. Nodes with no background indicate new technical contributions that are not yet completed. Dark green nodes indicate work that is completed, and dark gray nodes indicate required resources that are not contributions of this thesis.
5 The Guess Calculator Framework

Our tool for studying password policies is the “guess calculator,” used as part of a framework for calculating guess numbers for passwords of a target policy. The framework incorporates a probabilistic context-free grammar that builds on the grammar developed by Weir et al. [86]. We first introduced the guess calculator in the paper “Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms” [46] and much of the proposed work in Section 7 involves enhancements to the framework. In this section I present the results of that paper after describing the guess calculator framework in detail.

The guess calculator framework can be used to evaluate password policies under a limited-knowledge, offline-attack threat model. In this threat model, the attacker has a file of hashed passwords. To identify passwords, the attacker must generate guesses, hash them, and compare the resulting hashes to the hashed passwords in the file. The attacker can make a large number of guesses, over 1 trillion, but does not know the true distribution of passwords of the target. We believe this threat model is realistic—many websites have recently had hashed password files stolen [22, 47–49, 55, 64, 71, 75, 80, 83], professional password crackers use tools that can make 350 billion guesses per second for certain hashing algorithms [34] and, as explained earlier, the lack of data for most password distributions makes them very difficult to characterize.

5.1 Previous approaches

Weir et al. studied various aspects of password policies in their 2010 paper, “Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords” [87]. This paper made major steps forward in the science of evaluating password policies under an offline-attack threat model, and this thesis expands on many of its concepts.

Figure 5 shows a high-level overview of previous approaches to generating guess numbers for a test set, based on the methodology of Weir et al. A similar approach was followed by Dell’Amico et al. with other password guessing algorithms [25]. First, input datasets of leaked passwords and cracking
Weighted datasets

- Leaked password sets
- Cracking dictionaries

Prune passwords to match target policy → Learn probabilistic context-free grammar → Enumerate and record guesses

For each password in the test set, assign it a guess number equal to its rank in the list of guesses.

Figure 5: Approach to guess number generation employed by Weir et al. [87]

dictionaries\(^1\) are collected and each dataset is assigned a weight based on how useful it is expected to be. This collection is pruned so that all input data conforms to the target policy. The data is then fed to an algorithm which learns a probabilistic context-free grammar, and a priority-queue algorithm is employed to generate a list of guesses in probability order. The guess number of a password is simply its rank in the list of guesses.

### 5.2 Improving on previous approaches

A downside to the approach of Weir et al. is the need to enumerate all guesses in probability order. This is a time-consuming process that limits the guess numbers that can be assigned. In Weir’s dissertation he examines guess numbers up to 1 billion [86], but modern cracking approaches can now make several hundred billion guesses per second [34]. A further limitation of this specific approach is in the quantity of training data that can be used. Due to implementation issues, Weir et al. were only able to train the grammar on a few million passwords. In contrast, our guess calculator has been trained on hundreds of millions of input elements, and can assign guess numbers beyond 50 trillion.

Figure 6 illustrates our guess calculator framework. It incorporates a number of improvements over previous approaches that are useful for evaluating password policies, as described below.

\(^1\)Sets of words designed to be used with cracking tools, also called “wordlists” [26].
Input datasets
- Leaked password sets
- Cracking dictionaries
- Spelling / inflection dictionaries
- Web corpus
- Password samples from target policy

Corpora of training data
- Structures
- Digits
- Special characters
- Alphabetic strings

Process and fork weighted combinations of datasets into four separate corpora

Prune structures to match target policy
Learn probabilistic context-free grammar

Generate lookup table
Parallel computation
Generate all patterns above given probability cutoff and record the number of guesses associated with each pattern

Parallel computation
Sort patterns by probability

Merge sorted patterns into single table
Using the number of guesses associated with each pattern, compute the prefix sum of each pattern in the sorted table

For each password in the test set, find its pattern in the lookup table. If the pattern is found, determine the rank of this password within the pattern, and add this rank to the prefix sum for the pattern to determine the password’s guess number. If the pattern is not found, assign the password a code that identifies which element of the password was not found.

Figure 6: Approach to guess number generation employed in the current version of the guess calculator framework [46]
Fine-grained specification of training data  Input datasets are processed into four different corpora used to train separate components of the grammar: structures, digits, special characters, and alphabetic strings. Each of the four corpora consist of a weighted combination of the input datasets, and structures are further pruned to match a target policy. This allows us to tailor guesses to match a complex policy while taking advantage of digits, symbols, and alphabetic strings from a larger corpus of passwords and dictionaries. For example, we can include all alphabetic strings from the Google Web Corpus [9].

Training on passwords from the target policy  Previous approaches to evaluating complex policies either did not have access to passwords created under such policies [87], or did not train on passwords from such policies to generate guess numbers [16, 38]. Our approach can include passwords from the target policy in our training sets to mimic an adversary that has access to a sample of passwords (acquired via phishing, for example).

Distributed computation of lookup table  A great deal of computation in our framework is parallelized. This is possible because we have restructured the generation of the lookup table to use a probability cutoff, rather than generating guesses in probability order. The “lookup table” itself stores patterns rather than explicit guesses, saving time and space when generating the table.\(^2\) Together, these improvements allow us to generate tables that can assign guess numbers in the trillions and beyond, in the same amount of time required to generate billions of guesses.

Cross-validation  Our implementation allows us to train on much larger sets than previous approaches, and our framework speeds up the process of generating guess numbers. Thanks to these gains, we can perform more careful policy evaluations. For example, we can perform cross-validation with samples from the target policy to generate guess number estimates in which we have more confidence. This use of cross-validation in passwords policy research has not previously been reported.

Uppercase nonterminal  We modified the grammar to account for uppercase characters in the set of nonterminals. This changes the set of nonterminals \(Q\) described in Section 3.3.1 into \(Q = \{ \{U, L\}^i, D_i, S_i\}.\) For example, where the

\(^2\)See Section 3.3.1 for the definition of “pattern” used here.
terminal “Password” would correspond to the nonterminal \(L_8\) in the original grammar, it corresponds to the nonterminal \((U_1L_7)\) in our framework. Weir et al. accounted for case in the set of terminals, instead of incorporating it into the grammar. By increasing the complexity of the grammar, our framework should be able to make more efficient guesses given sufficient input data.

A requirement of the guess calculator framework not shared by previous approaches is a need for plaintext passwords in the test set. Without plaintext passwords, it would not be possible to determine the pattern of a password to find it in the lookup table. One could modify the framework to output a list of guesses in probability order, but we do not consider such a scenario in this thesis.

### 5.2.1 Intelligent skipping algorithm

Generating only those patterns above a probability cutoff is itself a nontrivial operation. This is because even a simple training set can result in a grammar that can generate over \(10^{80}\) guesses. Traversing this massive search space is only possible due to the unique form of Weir’s PCFG and an algorithm we call “intelligent skipping.”

Each structure is processed independently. As explained earlier (Section 3.3.1), a structure is a string of nonterminals and each nonterminal is replaced solely by one of a set of terminals. Multiple terminals can share the same probability but here we assume without loss of generality that all terminals have unique probabilities.

We can index the patterns generated by a structure \(S\) using a mixed-radix number \(M\), with places \(M_{|S|}\) to \(M_1\), where \(|S|\) is the number of nonterminals in the structure. The base for each place \(M_i\) is the number of terminal replacements for that place. We donate this by \(#(M_i)\). A value of 0 for \(M_i\) indicates the highest probability terminal, and a value of \((#(M_i) - 1)\) corresponds to the lowest probability terminal.

For each structure, we start with \(M = 0 \ldots 00\). This corresponds to the highest probability pattern that can be produced by the structure. If the probability \(Pr(M)\) is above the cutoff, we output the pattern and compute \(M = M + 1\) using mixed-radix arithmetic. This continues until \(Pr(M)\) is below the cutoff. Now we want to skip past other patterns that are logically below the cutoff. For example, if \(M = 34502\), we skip to \(M = 34510\) because all intermediate patterns must be below the cutoff. For example, if \(M = 34502\), we skip to \(M = 34510\) because all intermediate patterns must be below the cutoff. After computing \(Pr(M = 34510)\), suppose that it is also below the cutoff. In this case, we could skip to \(M = 34520\) but we already know this
will be below the cutoff because the only change is \( M_2 = 1 \rightarrow 2 \) which is a lower probability terminal. By noting that \( M_1 = 0 \) when skipping, we can jump further ahead to \( M = 34600 \). If this is also below the cutoff, we can skip even further to \( M = 35000 \). Using this algorithm, we can iterate over the entire space of patterns while ensuring that no patterns above the cutoff are missed.

5.3 Results

We evaluated eight policies using the guess calculator framework. This section provides a brief overview of the results of that evaluation. Sample passwords were collected for each policy using Mechanical Turk; more details on our collection methodology can be found in our previous papers [46, 52]. The policies are described below, along with the instructions given to Mechanical Turk participants.

**basic8survey** Participants were given a scenario in which they were told, “To link your survey responses, we will use a password that you create below; therefore it is important that you remember your password.” The only requirement for this policy was that the “Password must have at least 8 characters.”

**basic8** Participants were given the hypothetical scenario:

Imagine that your main email service provider has been attacked, and your account became compromised. You need to create a new password for your email account, since your old password may be known by the attackers. Because of the attack, your email service provider is also changing its password rules. Please follow the instructions below to create a new password for your email account. We will ask you to use this password in a few days to log in again, so it is important that you remember your new password. Please take the steps you would normally take to remember your email password and protect this password as you normally would protect the password for your email account. Please behave as you would if this were your real password!

These instructions were used for all other policies after they were found to lead to more secure passwords than the basic8survey instructions. The only requirement for this policy was that the “Password must have at least 8 characters.”
**basic16** The only requirement for this policy was that the “Password must have at least 16 characters.”

**dictionary8** This policy required that the “Password must have at least 8 characters. It may not contain a dictionary word.” There are many ways to perform a dictionary check, and we opted to perform our check in the style of Carnegie Mellon’s Information Security Office, which governs our single-sign-on password system. Our check removes non-alphabetic characters and checks the remainder against a dictionary, ignoring case. We use a public cracking dictionary provided by openwall.com as our dictionary.

**comprehensive8** Participants were given the composition policy “Password must have at least 8 characters including an uppercase and lowercase letter, a symbol, and a digit. It may not contain a dictionary word.” We performed the same dictionary check as in dictionary8.

**blacklistEasy, blacklistMedium, blacklistHard** Our blacklist policies have a minimum length requirement of eight characters and perform a case-insensitive check of the entire user’s password against a blacklist. For blacklistEasy, we use a Unix dictionary as our blacklist. For blacklistMedium, we use a paid version of the openwall.com dictionary with 40M entries. For blacklistHard, we use Weir’s algorithm to create a list of five billion guesses to use as a blacklist.

### 5.3.1 Training

As presented in Figure 6 (page 12), we start training the guess calculator by processing input datasets into four corpora for structures, digits, special characters, and alphabetic strings. Datasets selected for each corpus for one experiment are shown in Table 1. Identical corpora are used for both digits and special characters. For expediency, this experiment uses the same corpora of training data for all eight policies, and only prunes structures for the basic16 policy. Samples from the target policies are multiplied so that their cumulative probability is equal to the cumulative probability of all other training data.

### 5.3.2 Policy findings

Figure 7 shows the performance of each policy as evaluated under the guess calculator framework. We present our results using “guessing curves,” a plot
Table 1: Datasets used for training the guess calculator.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Training sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structures</td>
<td>[MS, O] [500]</td>
</tr>
<tr>
<td>Digits</td>
<td>[MS, O, RY] [500]</td>
</tr>
<tr>
<td>Special characters</td>
<td>[MS, O, RY] [500]</td>
</tr>
<tr>
<td>Alphabetic strings</td>
<td>[MS, O, RY, W2, I] [500]</td>
</tr>
</tbody>
</table>

**Key:**
- MS – Myspace
- O – Paid Openwall dictionary
- RY – RockYou
- W2 – Webster’s Second International dictionary (public-domain)
- I – Inflection list from http://wordlist.sourceforge.net/

500 – 500 random sample passwords from each policy. The brackets indicate that these passwords are weighted so that their cumulative probability is equal to the cumulative probability of all other training data.

of the percentage of passwords cracked versus the number of guesses. Curves are limited to a “cutoff” number of guesses on the right side of the graph (50 trillion for this experiment) that corresponds to the probability cutoff selected when creating the lookup table. By examining guessing curves, we can discuss the relative strength or weakness of various policies. For example, we can see that basic16 has fewer passwords cracked than comprehensive8 from about $10^{11}$ guesses up to the cutoff, so we can say that it is stronger against a limited-knowledge attacker making a number of guesses in that range.

For a given guess number, we can also compare the proportions of passwords cracked for any two conditions with a $\chi^2$ or Fisher’s Exact test to see if the proportions are significantly different. At the cutoff, we find that basic16 is the strongest policy, followed by comprehensive8 and blacklistHard which are not significantly different. At one million or one billion guesses, however, both comprehensive8 and blacklistHard are significantly stronger than basic16.

Finding that basic16 is a stronger policy than comprehensive8 after a large number of guesses is important for two reasons. First, basic16 was considered less secure than comprehensive8 by our participants. Our participants were significantly more likely to agree with “If my main email account required me to change my password using the same requirements as used in this study, it would make my email account more secure” for comprehensive8 (67%) than for basic16 (57%) [52]. This impression extends to system administrators—NIST assigns both of those policies equivalent security but comprehensive8 was chosen over basic16
for Carnegie Mellon’s policy based on “common practice.” Second, basic16 is a more usable policy than comprehensive8 based on our usability evaluation [52]. This suggests that current policies are not optimized for both security and usability; there is room to move in both dimensions. This gives us some hope that through our policy evaluations we might discover additional policies that are more secure and more usable than common practice.

5.3.3 Methodological findings

The guess calculator framework can produce guess numbers in a mostly automated fashion given target passwords, training data, and a policy specification. However, the training data still needs to be selected and weighted, and this can have an impact on the resulting guess numbers. We performed further experiments with our data to measure this impact.

---

3In a personal communication, a representative of our Information Security Office stated, “Our guidance was developed based on common industry practice. For example, the Center for Internet Security, the NSA and Microsoft all recommend essentially the same 8 character + complexity policy. Many educational institutions employ similar policies. Absent strong evidence supporting alternatives, we stuck with common practice.”
Additional training samples  Figure 8 contrasts the effects of additional samples on the basic8 and comprehensive8 policies. For each policy, we plot a guessing curve using prior data only (P4, shown in black) plus 500 to 2,500 additional samples in increments of 500 (B8a–B8e, C8a–C8e). For basic8, additional samples from the target policy help slightly with early guesses but not in the long term. Compare this with comprehensive8, where each addition of 500 passwords increases the percent guessed by about 2% at the cutoff. Diminishing returns are reached at 2,000 passwords, however, as going from 2,000 to 2,500 passwords seems to provide no benefit. This suggests that additional samples from the target policy are more useful with complex policies than with simple policies, perhaps because much more data is available for simple policies from public sources.

Weighting training data  As mentioned above, samples were scaled so that their cumulative probability was equal to the cumulative probability of other training data. We also evaluated weightings of one-tenth and ten times the cumulative probability of other training data. Surprisingly, we found that weighting did not have much effect on guessing performance. Equal weighting slightly outperformed the other weightings at some points in the guessing curve, but no weighting was significantly better than other weightings at all points.

Comparing subsets  Finally, we examined the performance of passwords collected under the comprehensive8 policy to passwords collected under other policies that meet the comprehensive8 requirements. This examination is important for those studying complex policies who lack a source of samples from the target
Figure 9: Comparison of guessing performance for the comprehensive policy compared to passwords from all other policies that meet the comprehensive requirements. Passwords collected under the comprehensive policy were significantly easier to guess than passwords from the comprehensive subset.

Weir et al. use a subset of passwords created under a weaker policy as a proxy in their evaluation of strict policies [87].

Figure 9 shows the results of our examination. We find that passwords created under the comprehensive policy are significantly weaker than passwords found in a subset of other policies. We theorize that this might be due to selection bias: users who create strong passwords under a weak policy might construct stronger passwords than the general population, regardless of policy.

6 Measuring Password Guessability for an Entire University

In collaboration with our Information Security Office, we were provided with the unique opportunity to evaluate a complex password policy currently in use at our university. We negotiated an agreement that allowed us to run a number of experiments with the guess calculator framework to generate guessing curves, though we did not have direct access to the passwords themselves or their guess numbers. The experiments we selected compared passwords from the target policy to password subsets from leaked datasets and passwords collected from online studies. We also used regression analysis to discover demographic and behavioral
factors that are correlated with password strength. These results are omitted from this document and can be found in our paper [61].

This comparison of password sources is intended to help researchers select password sources for future research. Since this research was conducted on a single target policy, however, the general conclusions that can be drawn from it might be limited.

6.1 Policies

The current policy at Carnegie Mellon is similar to the comprehensive policy described in Section 5.3 except that it uses a smaller dictionary for the dictionary check. These accounts are part of a single-sign-on system that can be used to access email, tax and payroll statements, health information, grades, transcripts, course registration, and other restricted university resources. We were given indirect access to two password sets that we call CMUactive and CMUinactive. The CMUactive set had 25,459 passwords and denotes active CMU accounts. The CMUinactive set belongs to users whose accounts were deactivated after they left the university but which have not yet been deleted, and contains 17,104 passwords. Of these, 1,635 passwords do not conform to the current policy.

We compared the CMU sets to the following sets of passwords collected from online studies:

**MTsim** 1,000 passwords collected from an MTurk experiment designed to simulate CMU password creation as closely as possible, both in policy requirements and in the website design.

**MTcomp8** 1,000 passwords collected from MTurk, matching the comprehensive policy described in Section 5.3. The only difference between the requirements of this policy and MTsim is in the size of the dictionary used for the dictionary check.

We also compared our results with leaked datasets from five websites. In each case, we use a subset of the website passwords that meet CMU’s requirements. Where more than enough conforming passwords are available, we draw the test set at random. Three of these leaked sets were leaked in plaintext; the other two come from the subset of the original leak that was successfully cracked.
Corpus | Training sets
--- | ---
Structures | MS, O, RY‡, Y‡, C‡, S‡, G‡
Digits | MS, O, RY‡, Y‡, C‡, S‡, G‡
Special characters | MS, O, RY‡, Y‡, C‡, S‡, G‡
Alphabetic strings | MS, O, RY‡, Y‡, C‡, S‡, G‡, W2, I, GW

Key:
MS – Myspace
O – Paid Openwall dictionary
W2 – Webster’s Second International dictionary (public-domain)
I – Inflection list from http://wordlist.sourceforge.net/
GW – Google Web Corpus
RY – RockYou
Y – Yahoo! Voices
C – CSDN
S – Stratfor
G – Gawker
‡ – Remainder after removing 1,000 random sample passwords for test set

Table 2: Public datasets used for training the guess calculator.

**RYcomp8** 1,000 plaintext passwords from RockYou (42,496 conforming, 32,603,144 total).

**Ycomp8** 1,000 plaintext passwords from Yahoo! Voices (2,693 conforming, 453,488 total).

**CSDNcomp8** 1,000 plaintext passwords from the Chinese Developer Network (12,455 conforming, 6,428,285 total).

**SFcomp8** 1,000 cracked passwords from Strategic Forecasting, Inc., also known as Stratfor. (8,357 conforming, 804,034 total).

**Gcomp8** 896 cracked passwords from Gawker (896 conforming, 694,064 total). All eight characters long.

### 6.2 Training

We trained several guessing models in the course of this study, with training data for the baseline model shown in Table 2. Note that between the evaluations of
Section 5 and this study, several new datasets became available, allowing us to add much more training data into the model. We train on all but 1,000 passwords from each set and use the remaining passwords for testing. The 1,000 passwords chosen for each test are taken from the subset of each dataset that conforms to the CMUactive policy.

Adding to the baseline model, which we call the “limited-knowledge” attacker, we trained three other models. The “Public+MTsim” model adds 3,000 MTsim passwords to all training corpora. Similarly, the “Public+CMUactive” model adds 3,000 CMUactive passwords to all training corpora. Finally, we model an “extensive-knowledge” attacker by adding 20,000 CMUactive passwords and 15,000 CMUinactive passwords to all training corpora.

### 6.3 Results

Figure 10 shows the results of our evaluation with the limited knowledge and extensive knowledge models. While the RYcomp8 policy is surprisingly close to the target policy, the other four public datasets are much farther away than the MTsim or MTcomp8 policies. Ycomp8 and CSDNcomp8 exhibit similar behavior to the comprehensiveSubset discussed previously (Section 5.3.3), providing an overestimate of the security of the target policy. They are both significantly different from CMUactive under the extensive-knowledge model, and Ycomp8 is significantly different under the limited-knowledge model as well.$^4$

---

$^4$As compared with a $G^p$ test. Further details on our statistical methodology are in our paper [61].
We confirm the disutility of datasets of cracked passwords for policy evaluation—both Gcomp8 and SFcomp8 were much easier to guess than the target policy. While this might be an obvious finding, there are many sets of passwords that are released without provenance. This finding underscores the importance of having a set of reliable passwords for policy evaluation.

To test the utility of MTsim as a source of training data, we compare the Public+MTsim model to the Public+CMUactive model. The result is shown in Figure 11. The guessing curve for Public+MTsim is not significantly different from the curve for Public+CMUactive. This indicates that the MTsim dataset might be a good proxy for training passwords from the target policy.

Finally, we examine several other password properties across the datasets. Figure 12 shows how the average length and number of digits, symbols, and
we find that MTsim is much closer to CMUactive than other datasets. Surprisingly, passwords in each policy. This measure is often used in security metrics that are than CMUactive, but MTsim and MTcomp

does your password measure up?” papers [46, 82]. We focus on common elements of passwords and common strategies that users employ in response to constraints, using analysis methods that have not been previously reported. Key elements of this research are detailed below.

Table 3: Empirical probabilities for the 10 most popular passwords. Probabilities that are not significantly different from CMUactive for a given password rank are grayed out and marked with a † (Bonferroni-corrected χ² test, p < 0.05). All other policies are weaker than CMUactive, but MTsim and MTcomp8 are the closest on this metric.

7 Remaining Work

Sections 5 and 6 covered two of the technical contributions of this thesis. In this section, I discuss the three remaining contributions and an optional project that might be included in the thesis if time allows. Section 10 contains a proposed timeline.

7.1 Deep analysis

We are conducting an in-depth, qualitative and quantitative study of the structure of passwords. This study is based on examining the several thousand passwords that we’ve collected so far, including data from both the “Guess again” and “How does your password measure up?” papers [46, 82]. We focus on common elements of passwords and common strategies that users employ in response to constraints, using analysis methods that have not been previously reported. Key elements of this research are detailed below.
Table 4: This table demonstrates how tokenization can find common semantic elements within passwords. The top five passwords containing the substring “love” from the RockYou dataset are shown along with their ranking in the overall dataset and the result of running these passwords through our tokenizer.

<table>
<thead>
<tr>
<th>Password</th>
<th>Ranking</th>
<th>Tokenization</th>
</tr>
</thead>
<tbody>
<tr>
<td>iloveyou</td>
<td>5</td>
<td>i love you</td>
</tr>
<tr>
<td>lovely</td>
<td>15</td>
<td>lovely</td>
</tr>
<tr>
<td>iloveu</td>
<td>22</td>
<td>i love u</td>
</tr>
<tr>
<td>loveme</td>
<td>38</td>
<td>love me</td>
</tr>
<tr>
<td>loveyou</td>
<td>52</td>
<td>love you</td>
</tr>
</tbody>
</table>

**Semantic elements**  Passwords appear to be composed of semantic elements, or *chunks* for short, and tokenizing passwords based solely on character class does not reveal these chunks. For example, the string “love” is very common in passwords, but the current guessing model is not powerful enough to learn this fact because it parses passwords based solely on character class. Table 4 shows the five most frequent passwords containing “love” from the RockYou dataset. All of the passwords in Table 4 are treated as single tokens by current parsers, because they have no way to break them down further. Using a tokenizer we developed, we can find common chunks in passwords and measure their frequency. While similar results have been reported before, we provide more details than previous work on the frequency and types of chunks used in passwords, and extract them with a transparent and automated process that uses the public Google Web Corpus [9].

Our findings suggest the incorporation of a tokenizer into our guessing model to identify and learn common chunks. We also find that chunks in a given password often exhibit dependencies on one another. This suggests that a more sophisticated grammar has the potential to model passwords better than Weir’s PCFG.

**Leetification study**  “Leetification” is the substitution of digits or symbols for letters in a word. Users might employ this strategy when creating passwords to improve their strength or comply with policies that require digits and symbols. We propose using Mechanical Turk workers to “unleetify” passwords from our datasets. Unleetification, as used here, is the conversion of digits or symbols in passwords to letters if this would reveal previously obscured words. For example, pa$$word should be unleetified to password, and love4all should be unleetified to loveforall. Once collected, this data could be used to derive leet-substitution rules.
and assign probabilities to them. In these examples, the derived leet-substitution rules are $s \rightarrow \$ and $for \rightarrow 4$. A more accurate understanding of leet-substitution could be used in the future for more accurate password modeling.

While leetification has been reported on previously [43, 91], it was limited to substitution rules chosen by the researchers. Our preliminary attempts to replicate this approach found that it misclassifies many passwords. In contrast, our proposed approach uses human intelligence to classify and code passwords. The study will run in two phases:

**Phase one**
Sample passwords are sent to Mechanical Turk for unleetification. Upon completion by workers, the results are analyzed for accuracy by members of our research group and the best workers are identified.

**Phase two**
The rest of our passwords are assigned to the best workers identified in Phase one, with periodic checks on accuracy. Each password is assigned to three different workers so inter-rater reliability can be assessed.

**Pauses and suggestions** When we collect passwords from Mechanical Turk, we also collect keystroke timing data during password creation and recall. We analyzed this data after tokenizing our passwords for semantic elements and found a significant correlation between chunks identified by tokenizing and chunks identified by inter-keystroke timing. In other words, users sometimes pause slightly between chunks when creating passwords.

This analysis is more interesting when performed on the “How does your password measure up?” dataset [82]. In this data, participants saw password meters accompanied by suggestions while creating their passwords. We recorded the suggestions shown to participants along with timing data. Our preliminary analysis suggests that password meter suggestions are correlated with the next character during pauses between chunks, but are not significantly correlated with user behavior when the user is in the middle of a chunk. If further analysis supports this correlation, it would provide an insight into user behavior and have implications for the security of policies that provide real-time suggestions during password creation. If users are more likely to incorporate suggestions between chunks in a password, an adversary with knowledge of the suggestion algorithm might be able to use this knowledge to improve guessing.
7.2 Improvements to the guessing model

We can make a number of improvements to our guessing model to improve its guessing efficiency. Making these improvements will give us a more reliable metric for evaluating samples of passwords created under various policies. We propose three major improvements: 1) incorporating a tokenizer into the framework, 2) learning string frequencies, and 3) using a more powerful grammar.

To effect these improvements on the guessing model, it is necessary to make changes to the guess calculator framework. An overview of the proposed design is shown in Figure 13, a modification of the design shown in Figure 6 on page 12. Nodes with planned improvements are shown with highlighted borders.

7.2.1 Implementation

Our current codebase is built on top of code originally released by Matt Weir under a GPLv2 license.\(^5\) This code was heavily modified to generate patterns above a particular probability, rather than guesses in probability order, using the intelligent skipping algorithm described in Section 5.2.1. It was also modified to incorporate an uppercase nonterminal and remove various implementation limitations.\(^6\) The guess calculator framework is built around this core code and includes functions for processing and parsing the input datasets into four corpora based on weighting parameters, pruning structures to match the target policy, sorting patterns, constructing the lookup table, and parsing passwords from the test set so they can be found in the lookup table.

Details on the implementation of the three improvements are given below. Implementation will be performed in two phases, corresponding to the first two improvements and the third improvement. The first two improvements do not require a change to the underlying grammar. Weir’s PCFG already supports string probability rules and structures with multiple nonterminals of the same class, though these features are not utilized in the original code. For example, the password *iloveyou* is currently parsed as the structure \(L_8\), because the current parser has no way to break alphabetic strings. By incorporating a tokenizer into the parser, the same password can be parsed as \(L_1L_4L_3\) without much change to

\(^5\)http://www.gnu.org/licenses/gpl-2.0.html

\(^6\)For example, guess numbers can overflow a long integer, so the GNU Multiple Precision Arithmetic Library is used and many integer operations are replaced with their respective operations from the library.
### Input datasets
- Leaked password sets
- Cracking dictionaries
- Spelling / inflection dictionaries
- Web corpus
- Password samples from target policy

### Corpora of training data
- Structures
- Digits
- Special characters
- Alphabetic strings

### Process and fork weighted combinations of datasets into four separate corpora

### Prune structures to match target policy

### Tokenize structures and alphabetic strings

### Learn more powerful grammar

### Quantize terminal probabilities

### Generate lookup table
- Parallel computation
  - Generate all patterns above given probability cutoff and record the number of guesses associated with each pattern
- Parallel computation
  - Sort patterns by probability
- Merge sorted patterns into single table
- Using the number of guesses associated with each pattern, compute the prefix sum of each pattern in the sorted table

### Assign guess numbers
- Map each password in the test set to matching structures in the grammar, ignoring tokenization
- Parallel computation
  - Assign guess number or code to each possible tokenization of each password
- For each password in the test set, assign the lowest guess number or most relevant code across all tokenizations

---

**Figure 13**: Proposed design for improved guess calculator framework.
the grammar-related code. This means we can make incremental improvements to modules of the framework while continuing to use it.

After the first two improvements are made and the modular structure of the framework has been refined, we will begin making changes to the underlying grammar. Some potential grammars are discussed in Section 8.4.1. The goal of upgrading the grammar is to better capture dependencies between elements, so phrase-based passwords like *iloveyou* are guessed as a phrase rather than an independent set of words.

**Incorporating a tokenizer**

We developed a local tokenizer using the Google Web Corpus (GWC) as a dataset. The GWC was published in 2006 and contains “English word n-grams and their observed frequency counts” [9] based on a corpus of one trillion words from web pages. Only n-grams with frequencies above 40 were kept by the publishers and the maximum n-gram length is five.

Our tokenizer is built on two data structures produced from processing the GWC. Processing begins by removing all non-alphabetic n-grams, then downcasing and aggregating duplicates from the remaining n-grams. We assume that words in passwords may not have the same case as they would on a webpage, so we ignore case when tokenizing. For each n-gram, we create a mapping from its untokenized version (removing spaces between tokens) to the tokenized version, and collect the results in a sorted table called a *token table*. Thus, for untokenized text for which at least one n-gram exists in the GWC, we can look up the text in our token table and examine the frequencies of all tokenizations. Our token table contains 1.35 billion rows.

However, it is necessary to tokenize text which does not appear in the GWC. This is because longer passwords, such as our basic16 passwords, might contain more than five tokens and go untokenized. This would make them much more difficult to guess unless the alphabetic strings in the password are found whole in the training data. In order to tokenize text which does not appear in the GWC, we use a greedy algorithm. The algorithm is built on a prefix tree, also called a trie, that is populated with the untokenized strings from the token table. Tries are optimized for performing prefix searches: given an arbitrary string, we can quickly find the longest prefix of that string in the trie.

Using the token table and trie structures, the algorithm for tokenizing a string is as follows. The longest prefix of the string is found using the trie, and this prefix is tokenized with the token table, using the most likely tokenization. If
Table 5: Example of our tokenizer operating on the phrase “correcthorsebatterystaple” which does not appear in the Google Web Corpus. Accepted tokens are shown in bold. After each iteration, the accepted tokens are removed from the string’s prefix and added to the final tokenization.

<table>
<thead>
<tr>
<th>Current string</th>
<th>Tokenized prefix</th>
<th>Tokenized string</th>
</tr>
</thead>
<tbody>
<tr>
<td>correcthorsebatterystaple</td>
<td>correct horse</td>
<td>correct</td>
</tr>
<tr>
<td>horsebatterystaple</td>
<td>horse battery</td>
<td>correct horse</td>
</tr>
<tr>
<td>batterystaple</td>
<td>battery</td>
<td>correct horse battery</td>
</tr>
<tr>
<td>staple</td>
<td>staple</td>
<td>correct horse battery staple</td>
</tr>
</tbody>
</table>

the tokenization has more than one token, only the first token is accepted. The accepted token is then removed from the prefix of the string, and the remainder is tokenized in the same manner. Thus, all but the first token of the previous tokenization is available to provide context for the next iteration. An example tokenization is shown in Table 5.

**Learning string frequencies and quantization**

The current guessing model does not learn alphabetic string probabilities from training data. Instead, it assigns all alphabetic terminals a uniform probability. This is described by Weir as an “implementation convenience” and was likely required when producing guesses in probability order. This is unnecessary with our guess calculator framework because we can produce patterns above a particular cutoff out of order and sort them later.

Unfortunately, learning string frequencies still poses a significant challenge to the framework. Using precise string frequencies greatly reduces the number of guesses associated with a given pattern. While this should increase guessing efficiency, it reduces the advantage of the framework over generating guesses directly. This would make it infeasible to reach the realistically large number of guesses that adversaries can generate [34]. To counteract this, we propose adding a quantizer to the framework to reduce the number of unique terminal probabilities. This brings an accuracy penalty, but we hope this will be mitigated by the efficiency gains that come from more accurate string probabilities.

**Using a more powerful grammar**

The final improvement involves replacing Weir’s PCFG with a more powerful grammar that can learn dependencies. In other words, we want to implement a
grammar that can learn token sequences from training data and produce guesses that incorporate those same sequences. Section 8.4.1 covers some promising research directions in this area. To our knowledge, this type of approach has never been attempted with passwords and is expected to take five months of work.

### 7.2.2 Evaluation

These improvements can be evaluated with existing data. Since we assume that passwords are not being guessed optimally by the current calculator, we can evaluate improvements to the guessing model by comparing guessing curves before and after an improvement. If guessing curves shift to the left, this means that passwords are being guessed earlier and the improvement has positively affected guessing efficiency. For each improvement, we can also produce a large amount of diagnostic data. For example, we can examine the actual guesses being made or see why certain passwords are never guessed.

Of particular importance to this evaluation are long passwords: passwords with 12 or more characters. Since our previous results suggest that a length requirement can be stronger for a large number of guesses than character class requirements, we want to have confidence that our guessing model is appropriate for long passwords. Outside of this thesis, we plan to collect several thousand long passwords. These passwords can then be used for testing our models, though their full evaluation is not part of this thesis.

### 7.3 Methodological guidance

As explained in Section 5.3.3, generating guess numbers for a given test set is mostly an automated process with the guess calculator framework. However, training data still needs to be selected and weighed, and proxy test sets might be required if passwords created under the target policy are not available.

Our goal is to compile the methodological lessons that we have learned into a single document to guide those who wish to evaluate password policies. The areas in which we will offer guidance are:

**Selection of training data** There are many possible configurations of training data, and our current approach to training data selection has been mostly arbitrary. Choosing which datasets should be included in training data and deciding how to weight them are required steps in using the guess
calculator framework, but we do not have empirical data to aid in making these decisions.

**Selection of test data** When passwords in use at an organization are not available, various proxies could be chosen as a test set. Our evaluation of Carnegie Mellon passwords found that passwords from online studies might serve as a reasonable proxy, while password subsets from leaked datasets sometimes overestimate the security of the target policy. We will also provide guidance on acquiring passwords from online studies.

**Setting parameters** The guess calculator framework has a few parameters that can be set such as the probability cutoff, tokenizer, and quantizer parameters. We will provide experimental results for various values.

In all cases, experimental data will be presented to support the guidance we provide. To that end, we will conduct a number of experiments with existing test sets and many different configurations of training data to understand how these changes can affect a policy evaluation.

### 7.4 Optional: Understanding mobile passwords

A way to tie together the parts of this thesis is to use the new calculator to conduct a security and usability analysis of previously unstudied policies. Given the rate of adoption of mobile devices as a primary computing device and the unknown effect of soft keyboards on password composition, examining passwords created on mobile devices seems like a natural choice.

The evaluation would proceed as follows: Select a small set of requirements to be deployed on mobile devices. These might include the same requirements evaluated earlier in this thesis, incorporated into a mobile policy. We can collect passwords created under these policies from Mechanical Turk by specifically targeting mobile device users and modifying our study infrastructure to gather data about participants’ devices. Once passwords are collected, we will conduct an in-depth analysis of them using the deep analysis techniques of Section 7.1, the improved calculator of Section 7.2, and the methodology of Section 7.3.

Adding this chapter to the thesis is contingent on completing the other chapters sooner than expected. In particular, implementing a more powerful grammar could take less than the five months that are currently reserved for it. If this component takes significantly less time, then the mobile password evaluation
described above will be added to the thesis as a final chapter. If not, the evaluation will be conducted separately so that it does not block completion of the thesis.

8 Overview of Related Work

Related work for this thesis includes research into password composition, password strength metrics, methodology, and modeling passwords with formal grammars.

8.1 In-depth analysis of password composition

Information about the properties of passwords seems to come from two types of sources: leaked datasets and surveys. In cases where the original passwords come from similar policies, analyses of leaked data and surveys tend to discover similar trends. In this section, I organize previous work on the composition of passwords based on the policy of the passwords being studied: simple or complex. We call a policy simple if it requires a length of 8 characters or less and at most one special character, and complex if its requirements are more strict.

It should be noted that one should only draw conclusions about password distributions from sets where all frequency information is retained. For example, after the LinkedIn breach, the file of hashes that was publicly released did not include the 57% of passwords that had already been cracked by the original attackers [41]. Though some information was left in the file to allow researchers to reconstruct some of the common passwords, this is a manual process. Any inferences about LinkedIn passwords made from only the released hashes would ignore the weakest and most frequent passwords in the distribution. This can lead to an underestimate of the policy’s guessability.

8.1.1 Simple policies

To date, there has not been a password leak from an organization with a complex policy, though many leaked password sets from commercial websites with simple policies have been studied. As investigated by Florêncio and Herley, commercial websites often have simple password policies [32] so the abundance of data from simple policies is not unusual. Sets associated with different websites tend to have many passwords in common: password, password1, 123456, 12345789, iloveyou, princess, angel, and others [27,40,74].
Thirty percent of RockYou passwords consist of letters followed by numbers, and in both MySpace and RockYou, the most popular suffix digit is 1 [27,77]. Over time, we see a trend of increasing use of numbers, and sometimes symbols, in passwords. Zviran and Haga reported on a study of users at a U.S. Department of Defense installation in 1999, where passwords had no requirements. They found an average password length of six characters, with 14% including numbers and less than 1% including symbols [92]. Surveying students in an information systems course about their email passwords in 2006, Bryant and Campbell found an average length of eight characters with around 65% using numbers and 3% using symbols [11]. In a survey of healthcare workers in 2008, Medlin et al. found an average length of seven characters with 87% using numbers and 16% including symbols [62]. And in 2010, Korkmaz and Dalkilic analyzed 2,500 plaintext passwords from a Turkish university with no composition requirements and found that 73% included at least one digit, while only 1% included at least one symbol [53].

In an attempt to characterize the frequency distribution of passwords (ignoring the passwords themselves), Malone and Maher looked at a few leaked datasets and found that they almost follow a Zipfian distribution, but it is not truly Zipfian [57]. Similarly, Bonneau looked at the frequency distribution of 69 million Yahoo! passwords and found that he could fit a Sichel distribution to the data, but a Kolmogorov-Smirnov test rejected the hypothesis that the sample data was drawn from the modeled distribution [6].

Jakobsson and Dhiman studied how dictionary words are modified and used as part of passwords. They examined “leet” substitutions and found that less than 0.2% of passwords in various leaked password sets used leet substitutions [43].

8.1.2 Complex policies

There is far less previous work on complex policies.

In a survey of users who recently transitioned to a complex policy, we reported that participants often made passwords that exceeded minimum composition requirements, contrary to expectations [79]. We also found, however, that users often place required symbols, digits, and uppercase characters in predictable locations [52,79].

Zhang et al. studied how users modify passwords under a complex policy when a previous password expires, finding that up to 17% of new passwords can be cracked in under five guesses if the old password is known [91]. Common transformations included incrementing a number, replacing one symbol with
another, and moving digits from the end to the beginning of the password. The authors also examined single-character leet transformations, but do not report the proportion of users that employ them.

8.1.3 Linguistic elements

Many analyses of passwords have found that most passwords contain linguistic elements [15, 62, 76, 92]. Names, dictionary words, place names, and keyboard patterns have been found in password sets [40, 42]. Perhaps explaining the prevalence of linguistic elements in passwords, Carstens et al. found that passwords constructed from meaningful “chunks” were more memorable than passwords constructed using more traditional advice [18].

Bonneau and Shutova examined Amazon payphrases by taking advantage of the fact that users are restricted from selecting a payphrase that has already been selected by another user [8]. This allowed for querying the set of selected payphrases. The policy required two or more words and no digits or symbols, which likely encourages linguistic elements. They found that many users selected noun bigrams found in the British National Corpus and the Google Web Corpus, and that movie and book titles provided effective guessing dictionaries.

Tokenizing

_tokenizing_, as used in this thesis, is the process of breaking a password into linguistic elements. The particular application to passwords has not been studied, but related work exists in the field of natural language processing. Tokenizing has been studied in the context of Web URLs [20], segmentation of Chinese text [56], and breaking of compound words for machine translation [10]. The general approach is to develop a language model from existing data, where words are separated by spaces, and apply this model to find likely splits in unseparated text. A similar approach is applied in this thesis, largely influenced by the tools made available by Wang et al. [84] who base their models on data from the Bing web crawler.

8.2 Security metrics

The most comprehensive study of password strength metrics thus far was conducted by Joseph Bonneau for his Ph.D. thesis published in 2012 [5]. Bonneau
also proposes several new metrics, and a brief survey of these and other relevant metrics follows.

8.2.1 Password distributions and entropy

A common metric for key strength used in security research is entropy, a concept from information theory [78], but applying this concept to passwords has been problematic. An accurate measure of entropy requires knowledge of the entire probability distribution of passwords. Since the distribution has a very heavy tail, its true distribution is unknown—Bonneau found, after collecting about 69 million passwords, 42.5% were unique in the dataset [6]. Standard entropy estimates produced from a sample that incompletely reflects the distribution, even a sample of 69 million, are biased toward an underestimate of the true entropy [17].

Carlton suggests that the number of samples required to get an accurate estimate of entropy is related to the smallest probability in the underlying distribution. If the smallest observed probability is \( p_k \), then \( N \gg \frac{1}{p_k} \) is required [17]. Paninski quantified the required number of samples more rigorously, and found that a reasonable estimate can found with a number of samples based on the number of “categories” in the underlying distribution [69]. For passwords, this would be the cardinality of the set of possible passwords. With \( m \) categories, Paninski suggests \( \frac{m^2}{(\log m)^2} \) samples. In either case, this appears to be far more samples than is feasible to collect. Using Paninski’s formula and Bonneau’s identification of 33,995,873 distinct passwords in his sample, around four trillion samples are required. This is likely an underestimate of the space of possible passwords, making the true number of required samples higher still.

In 2010, our research group published an alternative method for estimating password entropy given a limited sample [52, 79]. Rather than estimate entropy based solely on the probability distribution of whole passwords, the algorithm measures the entropy of several features of each password and sums them to estimate the entropy of the distribution of the whole. In order to have enough samples to accurately estimate the entropy of the features, 1,000 samples from a given policy are recommended. This metric is not intended to provide an accurate measure of true password entropy. Rather, it is intended to provide an alternative metric that can be used to compare sets of passwords created under various policies with limited samples.

It should be noted that even with an accurate measurement, entropy might not be predictive of other security metrics. If the true metric of interest is guessing
difficulty, for example, or a probability measure for the weakest passwords of a policy, one can posit a family of distributions in which these measures are held constant while entropy is made arbitrarily large. Bonneau discusses this issue extensively [5, 6]. It is possible that real-world password distributions are constrained in some way that makes these theoretical results inapplicable, but this question has not been thoroughly studied. Recently, we provided some evidence that our entropy estimate might be more closely correlated with guessing difficulty than naïve metrics based solely on policy requirements, such as those proposed by NIST [46].

8.2.2 Metrics based on password cracking

For a given password and guessing algorithm, “guessability” measures the number of guesses needed by the algorithm to guess that password. This particular definition of guessability is parametrized by the guessing algorithm, which means that very different measures of guessability can be obtained when using different algorithms.

To avoid this problem, Bonneau characterizes guessability based on an adversary with perfect knowledge [6]. This obviates the details of the guessing algorithm. Examining the Yahoo! dataset that he collected, one estimates that after 200,000 guesses, 25% of passwords are guessed. In about 3,000,000 guesses, an adversary would guess 50% of passwords. However, there are some issues with these figures. Though it might be possible to derive such figures for a particular policy where a large amount of data has been collected, it is not possible in the case of complex policies whose distributions are unknown. A second and related point is that if an attacker cannot produce such efficient guesses in practice, then we are not accurately estimating the real-world security resulting from a given policy.

In 2009, Weir et al. presented the first password-cracking model in which guesses are made in probability order [88]. The authors found their technique to be more effective than John the Ripper, an open-source password cracking tool that does not base its guessing on probability [26]. In a separate study, Zhang et al. found Weir’s algorithm to be the most effective among the techniques they used [91]. Weir also made his tool and source code available online for other researchers [85].

Narayanan and Shmatikov introduced a password-cracking technique based on Markov models in 2005 [67]. While each guess of this algorithm is assigned a probability, the paper does not present an algorithm for producing guesses in
probability order. However, Marechal [59] and Weir [86] both examine this model and find it more effective than John the Ripper. Further, the paper introduces the idea of quantizing probabilities, which the authors call “discretizing,” to reduce memory usage at the cost of accurate password probabilities.

8.3 Methodology of password strength measurement

Having a good metric is not sufficient to estimate the guessability of passwords accurately. In this section, I discuss work related to methodological issues of password strength measurement.

8.3.1 Machine Learning

The basic approach used in this thesis for estimating the guessability of a policy is a subjective Bayesian approach [31]. It is not feasible to collect enough samples from a complex policy to estimate guessability using previous methods, but in a subjective Bayesian approach one combines new samples with prior data to generate estimates. The new data and prior data can be weighted as desired based on how informative the prior data is expected to be, or the appropriate weight can be selected based on the performance of a resulting model.

Methodological issues in model generation and estimation have been studied extensively in the machine learning literature [4, 29, 36, 51]. The best practices include:

**Holdout data**

Separate test data (on which the quantity is estimated) and training data (on which the model is trained).

**Cross validation**

Rotate training and test data throughout the sample so all data is used an equal number of times in training and testing.

**Randomization**

Both training and test sets should match the underlying distribution as closely as possible. With password datasets, the original data might be ordered in some non-random way, e.g., chronologically or alphabetically. Naively partitioning data of this form into training and test sets can produce samples that do not accurately reflect the underlying distribution.
Data collection and representative samples

A proper study of password policies requires a randomized experiment, in which participants are randomly assigned a password policy. Organizations could conduct such experiments, but it does not appear that an experiment of this type has been reported. Instead, password researchers have turned to conducting user studies in which participants create passwords used for the duration of the study.

However, this can create concerns of ecological validity. As we said in our 2011 paper “Of Passwords and People”:

It is difficult to demonstrate ecological validity in any password study where participants are aware they are creating a password for a study, rather than for an account they value and expect to access repeatedly over time. Ideally, password studies would be conducted by collecting data on real passwords created by real users of a deployed system.

Fahl et al. conducted a study on the ecological validity of passwords by comparing passwords created by users for a study to their real passwords [30]. They find a number of differences between the two datasets. They also find that participants who self-reported behaving differently for the study than they would in real life were more likely to create unrealistic passwords. The difficulty of studying passwords used for high-value accounts has consistently limited password research, and acquiring quality password samples from a complex policy is a non-trivial task.

Researchers have made various attempts to improve the validity of passwords created for user studies. In studying a new authentication mechanism, Karlof et al. made use of a financial incentive to add value to users’ accounts. They conducted a user study involving deception where participants’ compensation (from $20–$41) was stored in an account that users created for the study [45]. They recruited over 200 participants at a cost of over $4,000. This approach would be prohibitively expensive for larger-scale studies. In contrast, many user studies have asked participants to create passwords that protect simulated accounts, small monetary amounts, a chance to win an iPod in a raffle, or access to university course materials including homework and grades [21, 24, 54, 90].

In this thesis, we use a methodology we developed in which passwords are collected from Amazon’s Mechanical Turk using a hypothetical scenario. Participants are initially paid $0.55 to create passwords that will be needed several
days later to complete the study and obtain a small bonus payment. If they return, participants are paid $0.70. Participants are told:

Imagine that your main email service provider has been attacked, and your account became compromised. You need to use a new password for your email account, since your old password may be known by the attackers.

Because of the attack, your email service provider is also changing its password rules. Instead of choosing your own password, one will be assigned to you. We will ask you to use this password in a few days to log in again so it is important that you remember your new password. Please take the steps you would normally take to remember your email password and protect this password as you normally would protect the password for your email account. Please behave as you would if this were your real password!

We compared this wording to other forms of instructions, and found that passwords created with this wording were significantly stronger than those created with a wording that did not contain a role-playing scenario [52]. The passwords of those who returned were also compared with those who did not, and no significant differences were found.

Many researchers have examined the use of Mechanical Turk workers as participants in human-subjects research. Buhrmester et al. found that American workers are slightly more representative of the U.S. population than other types of internet samples, and significantly more diverse than samples used in typical lab-based studies that heavily favor college-student participants [12]. This study, and others, found that Mechanical Turk tasks with appropriate screening criteria can provide high-quality user-study data [12, 28, 50, 70, 81].

It is also possible to ignore new data collection. Weir et al. used the RockYou set of leaked passwords to study complex policies by examining the subset of passwords from the set that conform to a complex policy [87]. One of the methodological contributions of this thesis is to examine the accuracy of such an approach.
8.4 Using formal languages to model passwords

As explained in Section 8.2.1, password distributions are heavy-tailed and are usually dominated by “singletons,” passwords that only appear once in the data set. A useful guessing algorithm must be able to produce guesses that do not appear in training data, otherwise singletons will never be guessed.

Crack, the first Unix password cracker [89], had a rich language for specifying “mangling rules.” These were rules that could be applied to words from an input dictionary to generate new guesses [66]. Marechal, one of the developers of John the Ripper, claims that modern password crackers derive their mangling-rule language from Crack [60].

Both Narayanan et al. and Weir et al. proposed probabilistic models of guessing [67, 88]. In these models, many guesses can be assigned probability values including guesses that do not appear in the training data. When these guesses are made in probability order, these models can provide a significant advantage in guessing efficiency over traditional password crackers [88].

The models used in previous work are still relatively crude, however. A first-order, character-level Markov model, as proposed by Narayanan and Shmatikov [67], will rarely generate intelligible text.\footnote{For an example of this, see [37].} Though higher-order Markov models might hold promise, the performance of such models has not been reported. In contrast, probabilistic context-free grammars (PCFGs) have been used previously with good results [88,91].

8.4.1 Extending probabilistic context-free grammars

If the grammar designed by Weir et al. were replaced, this might improve its guessing efficiency. In particular, the context-freeness of the grammar might be a hindrance in modeling passwords. Different elements of a password might be related to one another, but the current grammar treats these elements as independent. Researchers in computational linguistics have examined this flaw of PCFGs and various approaches have been developed to mitigate it. Early efforts in extending context-free grammars focused on context-sensitive grammars, more powerful grammars that can easily support the inclusion of contextual information [44]. However, parsing the strings of a context-sensitive language is computationally expensive, so researchers have moved on to extensions of PCFGs that still contain context-free rules yet can capture dependencies between elements.
These approaches are broadly called “lexicalized” grammars [23]. By adding a large number of non-terminals to the grammar, one for each tuple of dependent elements, dependencies can be captured while parsing remains computationally tractable. The most promising recent work in this field is the work of Petrov et al. [72, 73] called a Hierarchical State-Split Probabilistic Context-Free Grammar (HSSPCFG). HSSPCFGs use an iterative algorithm to steadily increase the number of non-terminals in the grammar until accuracy no longer improves.

Chapter Outline

After the introductory material, each of the technical contributions will be covered in a single chapter. The organization will be similar to the organization of this document. If time allows, the thesis might conclude with an evaluation of mobile passwords using the improved guess calculator. While not a technical contribution, this chapter would advance our understanding of mobile passwords and provide an example use of the guess calculator framework and our methodology.

Introduction This chapter will discuss the motivation for this research and briefly cover how this research differs from previous work.

Background and Related Work This chapter will cover various concepts necessary for understanding this thesis such as: password policies, threat models, and using formal languages to model passwords.

The Guess Calculator Framework This chapter consists of material from the “Guess again” paper [46]. It introduces our specific approach to measuring password strength: using a guess calculator with training and test data, guessing curves, and using Mechanical Turk to gather sample passwords. It also includes the results of our evaluation of eight password policies.

Guessability for an Entire University This chapter presents the results of our evaluation of the single-sign-on passwords used at Carnegie Mellon and a comparison of them with passwords collected from other sources.

Deep Analysis of Password Creation This chapter includes an in-depth lexical and linguistic analysis of passwords and the password-creation process, with insights into how often users engage in particular behaviors when creating passwords for various policies. The chapter concludes with a discussion of
how well these behaviors are accounted for in current password crackers, and proposed methods for modifying password crackers to account for the predictable behaviors that were identified.

**Improvements to the Guessing Model** This chapter describes improvements to our guessing model, and shows how previous analyses change when evaluated with the improved framework.

**Methodological Lessons in Password Strength Analysis** This chapter covers issues in methodology that arise when evaluating policies with the guess calculator framework. These issues include: test data selection when samples from the target policy are not available, training data selection in the same scenario, and choices for various parameters. Guidance will be based on guessing performance under various scenarios.

**Optional: Evaluation of Mobile Passwords** If time allows, an additional chapter will be added to the thesis in which we evaluate mobile passwords using the methodology and improved framework produced in this thesis. This evaluation will consist of a security and usability analysis of mobile passwords collected from Mechanical Turk compared with passwords created on hardware keyboards.
10 Timeline

<table>
<thead>
<tr>
<th>Task Description</th>
<th>Start Date</th>
<th>End Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thesis completion</td>
<td>May 2013</td>
<td>June 2014</td>
</tr>
<tr>
<td>Analysis of CMU single sign-on passwords</td>
<td>May 2013</td>
<td>August 2013</td>
</tr>
<tr>
<td>Improve tokenizer</td>
<td>June 2013</td>
<td>July 2013</td>
</tr>
<tr>
<td>Incorporate string frequencies</td>
<td>July 2013</td>
<td>September 2013</td>
</tr>
<tr>
<td>In-depth passwords analysis</td>
<td>August 2013</td>
<td>October 2013</td>
</tr>
<tr>
<td>Implement better grammar</td>
<td>October 2013</td>
<td>March 2014</td>
</tr>
<tr>
<td>Results analysis and write-up</td>
<td>March 2014</td>
<td>April 2014</td>
</tr>
<tr>
<td>Thesis write-up</td>
<td>April 2014</td>
<td>June 2014</td>
</tr>
<tr>
<td>Optional - mobile passwords study</td>
<td>June 2014</td>
<td>September 2014</td>
</tr>
</tbody>
</table>

Expected graduation date: Summer 2014
Bibliography


Christin, N., Cranor, L. F., and Lopez, J. 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, Oakland

Mashable (June 2012).

(Oct. 2011).

(June 2012).

[50] Kittur, A., Chi, E. H., and Suh, B. Crowdsourcing user studies with Mechanical

[51] Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation
and model selection. In International joint Conference on artificial intelligence,

Cranor, L. F., and Egelman, S. Of passwords and people: measuring the effect
of password-composition policies. In Proceedings of the SIGCHI Conference on
Human Factors in Computing Systems, CHI ’11, ACM (New York, NY, USA,
2011), 2595–2604.

[53] Korkmaz, I., and Dalkilic, M. E. The weak and the strong password
preferences: a case study on turkish users. In Proceedings of the 3rd international
conference on Security of information and networks, SIN ’10, ACM (New York,
NY, USA, 2010), 56–61.

[54] Kuo, C., Romanosky, S., and Cranor, L. F. Human selection of mnemonic
phrase-based passwords. In SOUPS ’06: Proceedings of the 2nd Symposium on

http://subinkushle.wordpress.com/2011/05/05/


[64] Molina, B. Sega reveals data breach affecting 1.3 million users. USATODAY.COM (June 2011).


[71] Peckham, M. Is valve’s steam hack as bad as the sony PlayStation debacle? Time (Nov. 2011).


[75] Rashid, F. Y. Anonymous breaches Booz Allen Hamilton to reveal 90,000 military passwords. eWeek (July 2011).


