Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms

Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy Vidas
Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio López
Carnegie Mellon University
Pittsburgh, PA, USA
{*pgage,sarangak,mmazurek,rshay,tvidas,lbauer,nicolasc,lorrie,julio.lopez*}*@cmu.edu*

***Abstract*—Text-based passwords remain the dominant authentication method in computer systems, despite significant advancement in attackers' capabilities to perform password cracking. In response to this threat, password composition policies have grown increasingly complex. However, there is insufficient research defining metrics to characterize password strength and using them to evaluate password-composition policies. In this paper, we analyze 12,000 passwords collected under seven composition policies via an online study. We develop an efficient distributed method for calculating how effectively several heuristic password-guessing algorithms guess passwords. Leveraging this method, we investigate (a) the resistance of passwords created under different conditions to guessing; (b) the performance of guessing algorithms under different training sets; (c) the relationship between passwords explicitly created under a given composition policy and other passwords that happen to meet the same requirements; and (d) the relationship between guessability, as measured with password-cracking algorithms, and entropy estimates. Our findings advance understanding of both password-composition policies and metrics for quantifying password security.**

***Keywords*-authentication; passwords; user study**

I. Introduction

Text-based passwords are the most commonly used authentication method in computer systems. As shown by previous research (e.g., [1]–[3]), passwords are often easy for attackers to compromise. A common threat model is an attacker who steals a list of hashed passwords, enabling him to attempt to crack them offline at his leisure. The many recent examples of data breaches involving large numbers of hashed passwords (Booz Allen Hamilton, HBGary, Gawker, Sony Playstation, etc.), coupled with the availability of botnets that offer large computational resources to attackers, make such threats very real [4]–[7]. Once these passwords have been cracked, they can be used to gain access not only to the original site, but also to other accounts where users reuse their passwords. Password reuse (exactly and with minor variations) is a common and growing practice as users acquire more online accounts [8], [9].

To mitigate the danger of such attacks, system administrators specify password-composition policies. These policies force newly created passwords to adhere to various requirements intended to make them harder to guess. Typical requirements are that passwords include a number or a symbol, that they exceed a certain minimum length, and that they are not words found in a dictionary.

Although it is generally believed that password-composition policies make passwords harder to guess, and hence more secure, research has struggled to quantify the level of resistance to guessing provided by different password-composition policies or the individual requirements they comprise. The two most commonly used methods for quantifying the effect of password-composition policies are estimating the entropy of the resulting passwords (e.g., [10], [11]), and empirically analyzing the resulting passwords with password-guessing tools (e.g., [12], [13]). The former, however, is not based on empirical data, and the latter is difficult to apply because of the dearth of available password sets created under different password-composition policies.

In this paper, we take a substantial step forward in understanding the effects of password-composition policies on the guessability of passwords. First, we compile a dataset of 12,000 plaintext passwords collected from different participants under seven different password-composition policies using an online study. Second, we develop approaches for calculating how long it would take for various password-guessing tools to guess each of the passwords we collected. This allows us to evaluate the impact on security of each password-composition policy.

Contributions. We make the following contributions:

1) We implement a distributed technique (*guess-number calculator*) to determine if and when a given password-guessing algorithm, trained with a given data set, would guess a specific password. This allows us to evaluate the effectiveness of password-guessing attacks much more quickly than we could using existing cracking techniques.
2) We compare, more accurately than was previously possible, the guessability of passwords created under different password-composition policies. Because of the efficiency of our approach (compared to guessing passwords directly), we can investigate the effective-

ness of multiple password-guessing approaches with multiple tunings. Our findings show that a password-composition policy requiring long passwords with no other restrictions provides (relative to other tested policies) excellent resistance to guessing.

3) We study the impact of tuning on the effectiveness of password-guessing algorithms. We also investigate the significance of test-set selection when evaluating the strength of different password-composition policies.
4) We investigate the effectiveness of entropy as a measure of password guessability. For each composition policy, we compare our guessability calculations to two independent entropy estimates: one based on the NIST guidelines mentioned above, and a second that we calculate empirically from the plaintext passwords in our dataset. We find that both measures of entropy have only very limited relationships to password strength as measured by guessability.

Mechanical Turk and controlled password collection. As with any user study, it is important to reflect on the origin of our dataset to understand the generalizability of our findings. We collected 12,000 plaintext passwords using Amazon's Mechanical Turk crowdsourcing service (MTurk). Many researchers have examined the use of MTurk workers (Turkers) as participants in human-subjects research. About half of all Turkers are American, with Indian participation increasing rapidly in the last 2-3 years to become about one third of Turkers [14]. American Turkers are about two-thirds women, while Indian Turkers are similarly weighted toward men [15]. Overall, the Turker population is younger and more educated than the general population, with 40% holding at least a bachelor's degree; both of these trends are more pronounced among Indian Turkers [14], [15].

Buhrmester et al. find that the Turker population is significantly more diverse than samples used in typical lab-based studies that heavily favor college-student participants [16]. This study, and others, found that well-designed MTurk tasks provide high-quality user-study data [16]–[19].

This analysis of MTurk has important implications in the context of studying passwords. We expect our findings will be more generalizable than those from lab studies with a more constrained participant base. Because we collected demographic information from our participants, our sample (and any biases it introduces) can be more accurately characterized than samples based on leaked password lists from various websites collected under uncertain circumstances.

A related consideration is that while our participants created real passwords that were needed several days later to complete the study and obtain a small bonus payment, these passwords did not protect high-value accounts. Password research has consistently been limited by the difficulty of studying passwords used for high-value accounts. Lab studies have asked participants to create passwords that protect simulated accounts, $5, a chance to win an iPod in a raffle, or access to university course materials including homework and grades [20]–[23]. Other studies have relied on leaked password lists like the RockYou set [13], [24]. While this set contains millions of passwords, it also contains non-password artifacts that are difficult to filter out definitively, its provenance and completeness are unclear, and it is hard to say how much value users place on protecting an account from a social gaming service. Other commonly used leaked password lists come from sites including MySpace, silentwhisper.net, and a variety of Finnish websites, with user valuations that are similarly difficult to assess [2], [25]. In Section VI, we briefly compare our MTurk users' behavior to results from a survey of people using higher-value passwords in practice.

Overall, although our dataset is not ideal, we contend that our findings do provide significant insight into the effects of password-composition policies on password guessability. Because so little is known about this important topic, even imperfect information constitutes progress.

Roadmap. In Section II we survey related work. We describe our data collection and analysis methodology in Sections III and IV. We convey our main results in Section V, and address their generalizability and ethical considerations in Section VI. We conclude in Section VII by discussing the implications of our results for future research and for defining practical password-composition policies.

II. Background and related work

Research on passwords has been active for many years. We first summarize the different types of data collection and analysis that have been used. We then discuss evaluations of the impact of password policies and metrics for quantifying password strength.

Collection and analysis of password data. Many prior password studies have used small sample sizes [26]–[29], obtained through user surveys or lab studies. Kuo et al. estimated the security of 290 passwords created in an online survey [21]. We also use an online survey, but we consider larger and more varied sets of passwords. In addition, we recruit participants using Mechanical Turk, which produces more diverse samples than typical lab studies [16].

Other studies analyze large samples of passwords ostensibly created by users for actual accounts of varying importance [1]–[3], [13], [30], [31]. Unlike these studies, we study the impact of different password policies on password strength and use passwords collected under controlled password-policy conditions.

Impact of password policies. Several studies have considered the impact of password policies on password strength. In lab studies, Proctor et al. [12] and Vu et al. [32] found passwords created under stricter composition requirements were more resistant to automated cracking, but also more difficult for participants to create and remember. We consider similar data for a much larger set of users, allowing for more comprehensive evaluation. Other findings suggest too-strict policies, which make creating and remembering passwords too difficult, induce coping strategies that can hurt both security and productivity [33]–[37]. Further, Florêncio and Herley found that the strictest policies are often used not by organizations with high-value assets to protect, but rather by those that do not have to compete on customer service [38].

An increasingly popular password-strengthening measure that we also investigate is subjecting new passwords to a blacklist check. Schechter et al. proposed a password policy in which passwords chosen by too many users are blacklisted for subsequent users [39]. This offers many theoretical advantages over other password-composition schemes.

Measuring password strength. Effective evaluation of password strength requires a proper metric. One possible metric is information entropy, defined by Shannon as the expected value (in bits) of the information contained in a string [40]. Massey connects entropy with password strength by demonstrating that entropy provides a lower bound on the expected number of guesses to find a text [41]. A 2006 National Institute of Standards and Technology (NIST) publication uses entropy to represent the strength of a password, but does not calculate entropy empirically [11]. Florêncio and Herley estimated theoretical entropy for the field data they analyzed [1].

An alternative metric of password strength is "guessability," which characterizes the time needed for an efficient password-cracking algorithm to discover a password. In one example, Weir et al. divide a large set of existing passwords into different categories based on composition, then apply automated cracking tools to examine how well NIST's entropy estimates predict measured guessing difficulty [13]. Castelluccia et al. use Markov models to measure password strength based on the distribution of already-selected passwords [42]. Dell'Amico et al. evaluate password strength by calculating guessing probabilities yielded by popular password-cracking heuristics [2]. We use a related approach but focus on comparing password policies.

Narayanan et al. discuss a password-cracking technique based on a Markov model, in which password guesses are made based on contextual frequency of characters [27]. Marechal [43] and Weir [44] both examine this model and find it more effective for password cracking than the popular password-cracking program John the Ripper [45]. Weir et al. present a novel password-cracking technique that uses the text structure from training data while applying mangling rules to the text itself [25]. The authors found their technique to be more effective than John the Ripper. In a separate study, Zhang et al. found Weir's algorithm most effective among the techniques they used [31].

In this work, we apply the Weir algorithm and a variation of the Markov model to generate blacklists restricting password creation in some of our study conditions, and to implement a new measure of password strength, the *guess number*, which we apply to user-created passwords collected under controlled password-composition policies.

III. Methodology: Data collection

In this section, we discuss our methodology for collecting plaintext passwords, the word lists we used to assemble the blacklists used in some conditions, and the eight conditions under which we gathered data. We also summarize participant demographics.

A. *Collection instrument*

From August 2010 to January 2011, we advertised a two-part study on Mechanical Turk, paying between 25 and 55 cents for the first part and between 50 and 70 cents for the second part. The consent form indicated the study pertained to visiting secure websites.

Each participant was given a scenario for making a new password and asked to create a password that met a set of password-composition requirements; the scenarios and requirements are detailed in Section III-C. Participants who entered a password that did not conform to requirements were shown an error message indicating which requirements were not met and asked to try again until they succeeded. After creating a password, participants took a brief survey about demographics and password creation. Participants were then asked to recall the password just created; after five failed attempts, the password was displayed. For the second part of the study, participants were emailed two days later and asked to return to the website and recall their passwords. We measured the incidence of passwords being written down or otherwise stored (via detecting browser storage and copy-paste behavior, as well as asking participants; see Section VI for details). The second part of the study primarily concerns memorability and usability factors. We report detailed results on these topics in a prior paper, which uses a large subset of the dataset we analyze here [46]; we briefly revisit these findings when we discuss our results in Section V.

B. *Word lists for algorithm training*

We use six publicly available word lists as training data in our analysis and to assemble the blacklists used in some of our experimental conditions. The *RockYou* password set [24] includes more than 30 million passwords, and the *MySpace* password set [47] contains about 45,000 passwords. (We discuss ethical considerations related to these datasets in Section VI.) The *inflection list*[1] contains 250,000 words in varied grammatical forms such as plurals and past tense. The *simple dictionary* contains about 200,000 words and is a standard English dictionary available on most Unix systems. We also used two cracking dictionaries from the Openwall Project[2] containing standard and mangled versions of dic-

[1] http://wordlist.sourceforge.net

[2] http://www.openwall.com/wordlists/

tionary words and common passwords: the *free Openwall list* with about 4 million words and the *paid Openwall list* with more than 40 million. While these data sources are not ideal, they are publicly available; we expect attackers would use these word lists or others like them for training data. In Section V-B, we consider the effect of a variety of training sets drawn from these word lists as well as our collected password data.

C. Conditions

Our participants were divided into eight conditions comprising seven sets of password-composition requirements and two password-creation scenarios. We used two scenarios in order to measure the extent to which giving participants different instructions affects password strength. The *survey scenario* was designed to simulate a scenario in which users create low-value passwords, while the *email scenario* was designed to elicit higher-value passwords. All but one condition used the email scenario.

In the *survey scenario*, participants 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."

In the *email scenario*, participants were told, "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!"

The eight conditions are detailed below.

basic8survey: Participants were given the survey scenario and the composition policy "Password must have at least 8 characters." Only this condition uses the survey scenario.

basic8: Participants were given the email scenario and the composition policy "Password must have at least 8 characters." Only the scenario differs from basic8survey.

basic16: Participants were given the email scenario and the composition policy "Password must have at least 16 characters."

dictionary8: Participants were given the email scenario and the composition policy "Password must have at least 8 characters. It may not contain a dictionary word." We removed non-alphabetic characters and checked the remainder against a dictionary, ignoring case. This method is used in practice, including at our institution. We used the free Openwall list as the dictionary.

comprehensive8: Participants were given the email scenario and 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. This condition reproduced NIST's comprehensive password-composition requirements [11].

blacklistEasy: Participants were given the email scenario and the composition policy "Password must have at least 8 characters. It may not contain a dictionary word." We checked the password against the simple Unix dictionary, ignoring case. Unlike the dictionary8 and comprehensive8 conditions, the password was not stripped of non-alphabetic characters before the check.

blacklistMedium: Same as the blacklistEasy condition, except we used the paid Openwall list.

blacklistHard: Same as the blacklistEasy condition, except we used a five-billion-word dictionary created using the algorithm outlined by Weir et al. [25]. For this condition, we trained Weir et al.'s algorithm on the MySpace, RockYou, and inflection lists. Both training and testing were conducted case-insensitively, increasing the strength of the blacklist.

These conditions represent a range of NIST entropy values: 18 bits for basic8 and basic8survey, 30 bits for comprehensive8 and basic16, and 24 bits for the four dictionary and blacklist conditions [11], [46]. We test the increasingly popular blacklist approach (see Section II) with a wide range of blacklist sizes.

D. Participant demographics

Of participants who completed part one of our study, 55% returned within 3 days and completed part two. We detected no statistically significant differences in the guessability of passwords between participants who completed just part one and those who completed both. As a result, to maximize data for our analyses and use the same number of participants for each condition, our dataset includes passwords from the first 1,000 participants in each condition to successfully complete the first part of the study. To conduct a wider variety of experiments, we used data from an additional 2,000 participants each in basic8 and comprehensive8.

Among these 12,000 participants, 53% percent reported being male and 45% female, with a mean reported age of 29 years. This sample is more male and slightly younger than Mechanical Turk participants in general [14], [16]. About one third of participants reported studying or working in computer science or a related field. This did not vary significantly across conditions, except between blacklistEasy and blacklistHard (38% to 31%; pairwise Holm-corrected Fisher's exact test [PHFET], $p < 0.03$). Participants in the basic16 condition were slightly but significantly older (mean

30.3 years) than those in blacklistHard, basic8, and comprehensive8 (means 28.6, 28.9, and 29.1 years respectively; PHFET, $p < 0.03$). We observed no significant difference in gender between any pair of conditions (PHFET, $p > 0.05$).

IV. Methodology: Data analysis

This section explains how we analyzed our collected password data. First, and most importantly, Section IV-A discusses our approach to measuring how resistant passwords are to cracking, i.e., guessing by an adversary. We present a novel, efficient method that allows a broader exploration of guessability than would otherwise be possible. For comparison purposes, we also compute two independent entropy approximations for each condition in our dataset, using methods described in Section IV-B.

A. Guess-number calculators

Traditionally, password guess resistance is measured by running one or more password-cracking tools against a password set and recording when each password is cracked. This works well when the exploration is limited to a relatively small number of guesses (e.g., 10^{10}, or roughly the number of guesses a modern computer could try in one day). However, as the computational power of potential adversaries increases, it becomes important to consider how many passwords can be cracked with many more guesses.

To this end, we introduce the *guess-number calculator*, a novel method for measuring guess resistance more efficiently. We take advantage of the fact that, for most deterministic password-guessing algorithms, it is possible to create a calculator function that maps a password to the number of guesses required to guess that password. We call this output value the *guess number* of the password. A new guess-number calculator must be implemented for each cracking algorithm under consideration. For algorithms (e.g., [13]) that use a *training set* of known passwords to establish guessing priority, a new *tuning* of the calculator is generated for each new training set to be tested.

Because we collect plaintext passwords, we can use a guessing algorithm's calculator function to look up the associated guess number for each password, without actually running the algorithm. This works for the common case of deterministic guessing algorithms (e.g., [13], [27], [43], [45]).

We use this approach to measure the guessability of a set of passwords in several ways. We compute the percentage of passwords that would be cracked by a given algorithm, which is important because the most efficient cracking tools use heuristics and do not explore all possible passwords. We also compute the percentage that would be cracked with a given number of guesses. We also use calculators to compare the performance of different cracking algorithms, and different training-set tunings within each algorithm. By combining guess-number results across a variety of algorithms and training sets, we can develop a general picture of the overall strength of a set of passwords.

We implemented two guess-number calculators: one for a brute-force algorithm loosely based on the Markov model, and one for the heuristic algorithm proposed by Weir et al., which is currently the state-of-the-art approach to password cracking [13], [31]. We selected these as the most promising brute-force and heuristic options, respectively, after comparing the passwords we collected to lists of 1, 5, and 10 billion guesses produced by running a variety of cracking tools and tunings. Henceforth, we refer to our implementations as the brute-force Markov (BFM) and Weir algorithms.

1) Training sets: Both algorithms require a *training set*: a corpus of known passwords used to generate a list of guesses and determine in what order they should be tried.

We explore a varied space of training sets constructed from different combinations of the publicly available word lists described in Section III-B and subsets of the passwords we collected. This allows us to assess whether complementing publicly available data with passwords collected from the system under attack improves the performance of the cracking algorithms. We further consider training-set variations specifically tuned to our two most complex policy conditions, comprehensive8 and basic16.

In each experiment we calculate guess numbers only for those passwords on which we did not train, using a cross-validation approach. For a given experiment, we split our passwords into n partitions, or *folds*. We generate a training set from public data plus $(n-1)$ folds of our data, and test it on the remaining fold. We use each of the n folds as test data exactly once, requiring n iterations of testing and training. We combine results from the n folds, yielding guess-number results for all of our passwords. Because training often involves significant computational resources, as described in Section IV-A3, we limit to two or three the number of iterations in our validation. Based on the similarity of results we observed between iterations, this seems sufficient. We describe our training and test sets in detail in Appendix A.

We do not claim these training sets or algorithms represent the optimal technique for guessing the passwords we collected; rather, we focus on comparing guess resistance across password-composition policies. Investigating the performance of guessing algorithms with different tunings also provides insight into the kind of data set an attacker might need in order to efficiently guess passwords created under a specific password-composition policy.

2) BFM calculator: The BFM calculator determines guess numbers for a brute-force cracking algorithm loosely based on Markov chains [27], [43]. Our algorithm differs from previous work by starting with the minimum length of the password policy and increasing the length of guesses until all passwords are guessed. Unlike other implementations, this covers the entire password space, but does not try guesses in strict probability order.

The BFM algorithm uses the training set to calculate the frequency of first characters and of digrams within the password body, and uses these frequencies to deterministically construct guessing order. For example, assume an alphabet of {*A, B, C*} and a three-character-minimum configuration. If training data shows that *A* is the most likely starting character, *B* is the character most likely to follow *A*, and *C* is the character most likely to follow *B*, then the first guess will be *ABC*. If the next-most-likely character to follow *B* is *A*, the second guess will be *ABA*, and so forth.

Our guess-number calculator for this algorithm processes the training data to generate a lookup table that maps each string to the number of guesses needed to reach it, as follows. For an alphabet of N characters and passwords of length L, if the first character tried does not match the first character of the target password, we know that the algorithm will try N^{L-1} incorrect guesses before switching to a different first character. So, if the first character of the password to be guessed is the k-th character to be tried, there will be at least $(k-1)N^{L-1}$ incorrect guesses. We can then iterate the computation: when the first character is correct, but the second character is incorrect, the algorithm will try N^{L-2} incorrect guesses, and so forth. After looking up the order in which characters are tried, we sum up the number of incorrect guesses to discover how many iterations will be needed before hitting a successful guess for a given password, without having to actually try the guesses.

3) Weir algorithm calculator: We also calculate guess numbers for Weir et al.'s more complex algorithm. The Weir algorithm determines guessing order based on the probabilities of different password *structures*, or patterns of character types such as letters, digits, and symbols [25]. Finer-grained guessing order is determined by the probabilities of substrings that fit into the structure. The algorithm defines a *terminal* as one instantiation of a structure with specific substrings, and a *probability group* as a set of terminals with the same probability of occurring.

As with the BFM calculator, we process training data to create a lookup table, then calculate the guess number for each password. The mechanism for processing training data is outlined in Algorithm 1. To calculate the guess number for a password, we determine that password's probability group. Using the lookup table created from the training set, we determine the number of guesses required to reach that probability group. We then add the number of guesses required to reach the exact password within that probability group. This is straightforward because once the Weir algorithm reaches a given probability group, all terminals in that group are tried in a deterministic order.

Because creating this lookup table is time-intensive, we set a cutoff point of 50 trillion guesses past which we do not calculate the guess number for additional passwords. This allows most Weir-calculator experiments to run in 24 hours or less in our setup. Using the structures and terminals learned from the training data, we can still determine whether passwords that are not guessed by this point will ever be guessed, but not exactly when they will be guessed.

Algorithm 1 Creation of a lookup table that, given a probability group, returns the number of guesses required for the Weir algorithm to begin guessing terminals of that group. An *l.c.s.* is a *longest common substring*, the longest substrings in a probability group made from characters of the same type. For example, for *UUss9UUU*, the *l.c.s.*'s would be *UU*, *ss*, *9*, and *UUU*. (In this example, *U* represents uppercase letters, *s* represents lowercase letters, and *9* represents digits.)

```
$\mathcal{T}$ = New Lookup Table
for all $structures\ s$ do
    for all $probability\_group\ pg \in s$ do
        for all $l.c.s. \in pg$ do
            $c_i$=Number of terminals of $l.c.s.$
            $p_i$=Probability of $l.c.s.$ in training data
        end for
        $probability = \prod p_i; size = \prod c_i$
        $\mathcal{T}.add$: $pg$, $probability$, $size$
    end for
end for
$Sort(\mathcal{T})$ by $probability$
Add to each value in $(\mathcal{T})$ the sum of prior $size$ values
```

Distributed computation. Calculating guess numbers for Weir's algorithm becomes data intensive as Algorithm 1 generates a large number of elements to build the lookup table $\mathcal{T}$. To accelerate the process, we implemented a distributed version of Algorithm 1 as follows. We split the top-most loop into coarse-grained units of work that are assigned to m tasks, each of which processes a subset of the structures in s. Each task reads a shared dictionary with the training data and executes the two internal loops of the algorithm. Each iteration of the loop calculates the probability and size for one probability group in s. This data is then sorted by probability. A final, sequential pass over the sorted table aggregates the probability group sizes to produce the starting guess number for each probability group.

We implemented our distributed approach using Hadoop [48], an open-source version of the MapReduce framework [49]. In our implementation, all m tasks receive equally sized subsets of the input, but perform different amounts of work depending on the complexity of the structures in each subset. As a result, task execution times vary widely. Nevertheless, with this approach we computed guess numbers for our password sets in days, rather than months, on a 64-node Hadoop cluster. The resulting lookup tables store hundreds of billions of elements with their associated probabilities and occupy up to 1.3 TB of storage each.

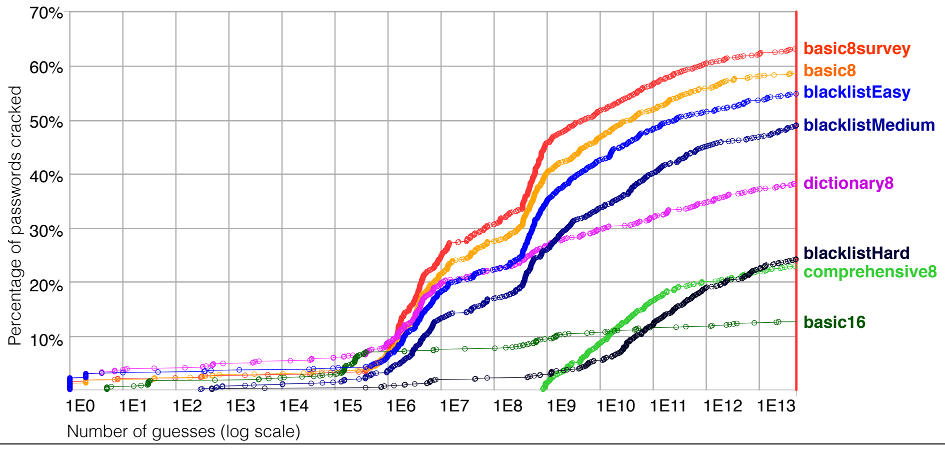

Figure 1. The number of passwords cracked vs. number of guesses, per condition, for experiment E. This experiment uses the Weir calculator and our most comprehensive training set, which combines our passwords with public data.

B. *Entropy*

To investigate how well entropy estimates correlate with guess resistance, we compare our guess-number results for each condition to two independently calculated entropy approximations. First, we apply the commonly used NIST guidelines, which suggest that each password-composition rule contributes a specific amount of entropy and that the entropy of the policy is the sum of the entropy contributed by each rule. Our second approximation is calculated empirically from the plaintext passwords in our dataset, using a technique we described previously [9]. In this method, we calculate for each password condition the entropy contributed by the number, content, and type of each character, using Shannon's formula [50]. We then sum the individual entropy contributions to estimate the total entropy of the passwords in that condition.

V. Findings

We calculated guess numbers under 31 different combinations of algorithm and training data. Although we do not have space to include all the results, we distill from them four major findings with application both to selecting password policies and to conducting password research:

- Among conditions we tested, basic16 provides the greatest security against a powerful attacker, outperforming the more complicated comprehensive8 condition. We also detail a number of other findings about the relative difficulty of cracking for the different password-composition policies we tested.
- Access to abundant, closely matched training data is important for successfully cracking passwords from stronger composition policies. While adding more and better training data provides little to no benefit against passwords from weaker conditions, it provides a significant boost against stronger ones.
- Passwords created under a specific composition policy do not have the same guess resistance as passwords selected from a different group that happen to meet the rules of that policy; effectively evaluating the strength of a password policy requires examining data collected under that policy.
- We observe a limited relationship between Shannon information entropy (computed and estimated as described in Section IV-B) and guessability, especially when considering attacks of a trillion guesses or more; however, entropy can provide no more than a very rough approximation of overall password strength.

We discuss these findings in the rest of this section. We introduce individual experiments before discussing their results. For convenience, after introducing an experiment we may refer to it using a shorthand name that maps to some information about that experiment, such as P for trained with public data, E for trained with everything, C8 for specialized training for comprehensive8, etc. A complete list of experiments and abbreviations can be found in Appendix A.

A. *Comparing policies for guessability*

In this section, we compare the guessability of passwords created under the eight conditions we tested. We focus on two experiments that we consider most comprehensive. In each experiment we evaluate the guessability of all conditions, but against differently trained guessing algorithms.

Experiment P4 is designed to simulate an attacker with access to a broad variety of publicly available data for training. It consists of a Weir-algorithm calculator trained on all the public word lists we use and tested on 1000 passwords from each condition. Experiment E simulates a powerful attacker with extraordinary insight into the password sets under consideration. It consists of a Weir-algorithm calculator trained with all the public data used in P4 plus 500

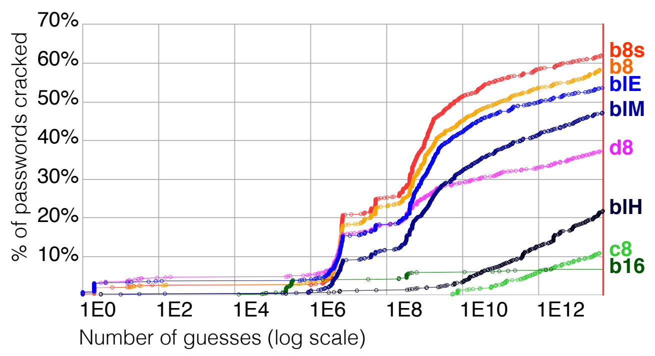

Figure 2. The number of passwords cracked vs. the number of guesses, per condition, for experiment P4. This experiment uses the Weir calculator and trains on a variety of publicly available data.

passwords from each of our eight conditions. We test on 500 other passwords from those conditions, with two-fold cross-validation for a total of 1000 test passwords. The results from these experiments are shown in Figures 1 and 2.

As these figures suggest, which password-composition policy is best at resisting guessing attacks depends on how many guesses an attacker will make. At one million and one billion guesses in both experiments, significantly fewer blacklistHard and comprehensive8 passwords were guessed than in any other condition.[3] At one billion guesses in experiment E, 1.4, 2.9, 9.5, and 40.3% of passwords were cracked in comprehensive8, blacklistHard, basic16, and basic8, respectively.

As the number of guesses increases, basic16 begins to outperform the other conditions. At one trillion guesses, significantly fewer basic16 passwords were cracked than comprehensive8 passwords, which were cracked significantly less than any other condition. After exhausting the Weir-algorithm guessing space in both experiments, basic16 remains significantly hardest to crack. Next best at resisting cracking were comprehensive8 and blacklistHard, performing significantly better than any other condition. Condition comprehensive8 was significantly better than blacklistHard in experiment P4 but not in experiment E. In experiment E, 14.6, 26.4, 31.0% of passwords were cracked in basic16, comprehensive8, and blacklistHard, respectively; in contrast, 63.0% of basic8 passwords were cracked.

Although guessing with the Weir algorithm proved more effective, we also compared the conditions using BFM. The findings (shown in Figure 3) are generally consistent with those discussed above: basic16 performs best.

In prior work examining memorability and usability for much of this dataset, we found that while in general less secure policies are more usable, basic16 is more usable than comprehensive8 by many measures [46]. This suggests basic16 is an overall better choice than comprehensive8.

[3]All comparisons in Sections V-A, V-B, and V-C tested using PHFET, significance level $\alpha = 0.05$.

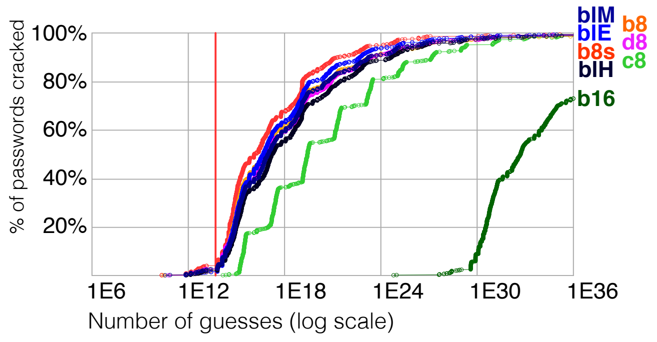

Figure 3. The number of passwords cracked vs. the number of guesses, using the BFM calculator trained on both our data and public data (B2). The red vertical line at 50 trillion guesses facilitates comparison with the Weir experiments. We stopped the Weir calculator at this point (as described in Section IV-A3), but because the BFM algorithm is so much less efficient, we ran it for many more guesses in order to collect useful data.

It is important to note that 16-character-minimum policies are rare in practice. Hence, current guessing algorithms, including the Weir algorithm, are not built specifically with them in mind. Although we do not believe this affects our overall findings, it may merit further investigation.

B. *Effects of training-data selection*

Like most practical cracking algorithms, the ones we use rely on training data to determine guessing order. As a result, it is important to consider how the choice of training data affects the success of password guessing, and consequently the guess resistance of a set of passwords. To address this, we examine the effect of varying the amount and source of training data on both total cracking success and on cracking efficiency. Interestingly, we find that the choice of training data affects different password-policy conditions differently; abundant, closely matched training data is critical when cracking passwords from harder-to-guess conditions, but less so when cracking passwords from easier ones.

For purposes of examining the impact of training data, the password-policy conditions we consider divide fairly neatly into two groups. For the rest of this section, we will refer to the harder-to-guess conditions of comprehensive8, basic16, and blacklistHard as *group 1*, and the rest as *group 2*.

Training with general-purpose data. We first measure, via three experiments, the effect of increasing the amount and variety of training data. Experiment P3 was trained on public data including the MySpace and RockYou password lists as well as the inflection list and simple dictionary, and tested on 1000 passwords from each of our eight conditions. Experiment P4, as detailed in Section V-A, was trained on everything from P3 plus the paid Openwall list. Experiment E, also described in V-A, used everything from P4 plus 500 passwords from each of our conditions, using two-fold cross-validation. Figure 4 shows how these three training sets affect four example conditions, two from each group.

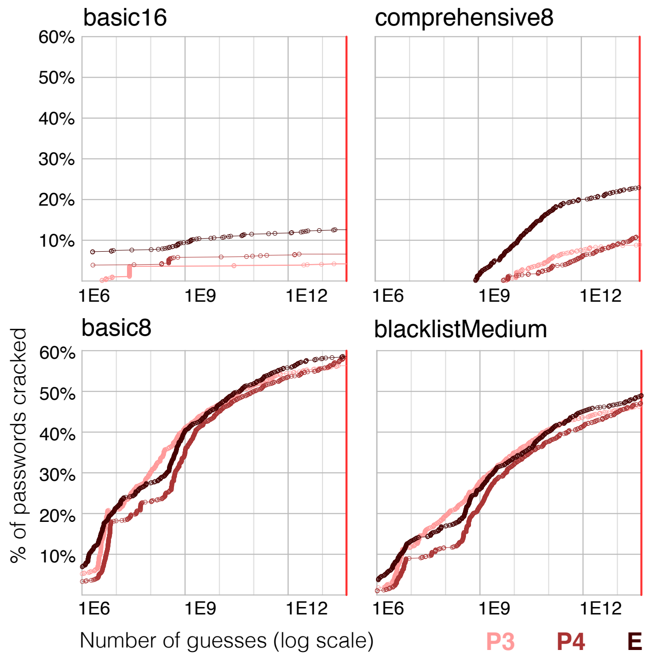

Figure 4. Showing how increasing training data by adding the Openwall list (P4) and then our collected passwords (E) affects cracking, for four example conditions. Adding training data proves more helpful for the group 1 conditions (top) than for the others (bottom).

As expected, cracking success increases as training data is added. For group 1, adding Openwall increases total cracking by 45% on average, while adding both Openwall and our data provides an average 96% improvement; these increases are significant for both experiments in all three conditions. In group 2, by contrast, the increases are smaller and only occasionally significant.

At one trillion and one billion guesses, the results are less straightforward, but increasing training data remains generally more helpful for cracking group 1 than group 2. Adding Openwall alone is not particularly helpful for group 1 conditions, with few significant improvements at either guessing point, but it actually decreases cracking at one billion guesses significantly for several group 2 conditions. (We hypothesize this decrease occurs because Openwall is a dictionary and not a password set, so it adds knowledge of structures and strings at the cost of accurately assessing their probabilities.) At these guessing points, adding our data is considerably more effective for group 1 than adding Openwall alone, increasing cracking for each of the three conditions by at least 50% (all significant). By contrast, adding our data provides little to no improvement against group 2 conditions at either guessing point.

Taken together, these results demonstrate that increasing the amount and variety of information in the training data provides significant improvement in cracking harder-to-guess conditions, while providing little benefit and sometimes decreasing efficiency for easier-to-guess conditions.

Training with specialized data. Having determined that training with specalized data is extremely valuable for cracking group 1 passwords, we wanted to examine what quantity of closely related training data is needed to effectively crack these "hard" conditions. For these tests, we focus on comprehensive8 as an example harder-to-guess condition, using the easier-to-guess basic8 condition as a control; we collected 3000 passwords each for these conditions.

In five Weir-algorithm experiments, C8a through C8e, we trained on all the public data from P4, as well as between 500 and 2500 comprehensive8 passwords, in 500-password increments. For each experiment, we tested on the remaining comprehensive8 passwords. We conducted a similar set of five experiments, B8a through B8e, in which we trained and tested with basic8 rather than comprehensive8 passwords.

Our results, illustrated in Figure 5, show that incrementally adding more of our collected data to the training set improves total cracking slightly for comprehensive8 passwords, but not for basic8. On average, for each 500 comprehensive8 passwords added to the training set, 2% fewer passwords remain uncracked. This effect is not linear, however; the benefit of additional training data levels off sharply between 2000 and 2500 training passwords. The differences between experiments begin to show significance around one trillion guesses, and increase as we approach the total number cracked.

For basic8, by contrast, adding more collected passwords to the training set has no significant effect on total cracking, with between 61 and 62% of passwords cracked in each experiment. No significant effect is observed at one million, one billion, or one trillion guesses, either.

One way to interpret this result is to consider the diversity of structures found in our basic8 and comprehensive8 password sets. The comprehensive8 passwords are considerably more diverse, with 1598 structures among 3000 passwords, as compared to only 733 structures for basic8. For comprehensive8, the single most common structure maps to 67 passwords, the most common 180 structures account for half of all passwords, and 1337 passwords have structures that are unique within the password set. By contrast, the most common structure in basic8 maps to 293 passwords, the top 13 structures account for half the passwords, and only 565 passwords have unique structures. As a result, small amounts of training data go considerably farther in cracking basic8 passwords than comprehensive8.

Weighting training data. The publicly available word lists we used for training are all considerably larger than the number of passwords we collected. As a result, we needed to weight our data (i.e., include multiple copies in the training set) if we wanted it to meaningfully affect the probabilities used by our guess-number calculators. Different weightings do not change the number of passwords cracked, as the same guesses will eventually be made; however, they can affect the order and, therefore, the efficiency of guessing.

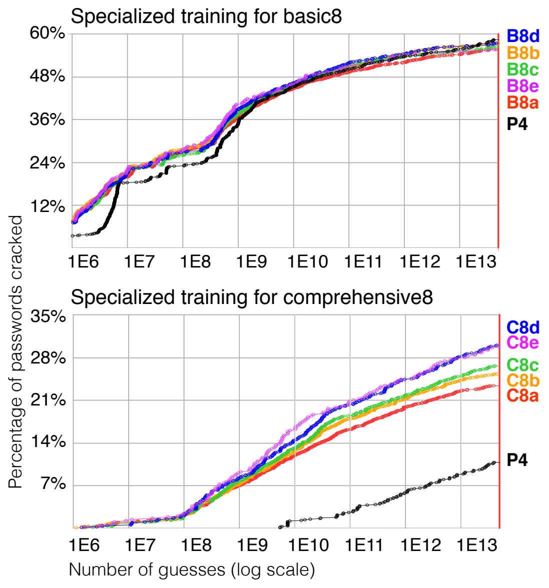

Figure 5. Top: Incremental increases in specialized training data have limited effect on the basic8 condition (B8a-B8e). Bottom: Incremental increases in specialized training data have a small but significant effect on the comprehensive8 condition (C8a-C8e). Results from P4 (the same public training data, no specialized training data) are included for comparison.

We tested three weightings, using 500 passwords from each condition weighted to one-tenth, equal, and ten times the cumulative size of the public lists. We tested each weighting on 500 other passwords from each condition.

Overall, we found that weighting had only a minor effect. There were few significant differences at one million, one billion, or one trillion guesses, with equal weighting occasionally outperforming the other two in some conditions. From these results, we concluded that the choice of weighting was not particularly important, but we used an equal weighting in all other experiments that train with passwords from our dataset because it provides an occasional benefit.

BFM training. We also investigated the effect of training data on BFM calculator performance, using four training sets: one with public data only, one that combined public data with collected passwords across our conditions, and one each specialized for basic8 and comprehensive8. Because the BFM algorithm eventually guesses every password, we were concerned only with efficiency, not total cracking. Adding our cross-condition data had essentially no effect at either smaller or larger numbers of guesses. Specialized training for basic8 was similarly unhelpful. Specialized training for comprehensive8 did increase efficiency somewhat, reaching 50% cracked with about 30% fewer guesses.

C. Effects of test-data selection

Researchers typically don't have access to passwords created under the password-composition policy they want to study. To compensate, they start with a larger set of passwords (e.g., the RockYou set), and pare it down by discarding passwords that don't meet the desired composition policy (e.g., [1], [13]). A critical question, then, is whether subsets like these are representative of passwords actually created under a specific policy. We find that such subsets are not representative, and may in fact contain passwords that are more resistant to guessing than passwords created under the policy in question.

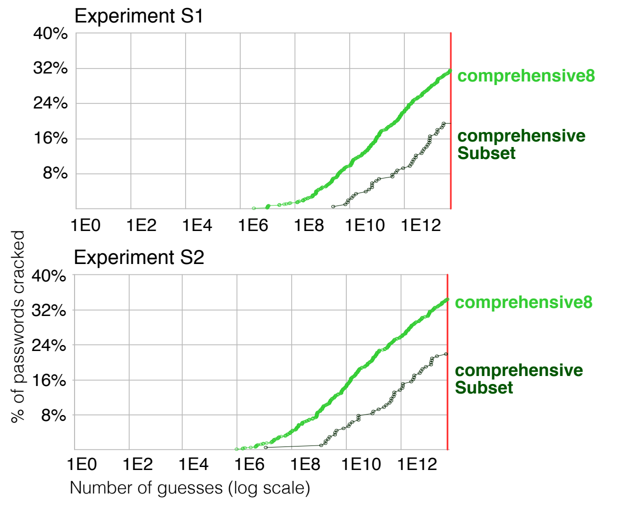

Figure 6. Passwords generated under the comprehensive8 condition proved significantly easier to guess than passwords that conform to the comprehensive8 requirements but are generated under other composition policies. In experiment S1 (top), the Weir calculator was trained with only public data; in experiment S2 (bottom), the Weir calculator was trained on a combination of our data and public data.

In our experiments, we compared the guessability of 1000 comprehensive8 passwords to the guessability of the 206 passwords that meet the comprehensive8 requirements but were collected across our other seven conditions (the *comprehensiveSubset* set). We performed this comparison with two different training sets: public data, with an emphasis on RockYou passwords that meet comprehensive8 requirements (experiment S1); and the same data enhanced with our other 2000 collected comprehensive8 passwords (experiment S2).

Both experiments show significant differences between the guessability of comprehensive8 and comprehensiveSubset test sets, as shown in Figure 6. In the two experiments, 40.9% of comprehensive8 passwords were cracked on average, compared to only 25.8% comprehensiveSubset passwords. The two test sets diverge as early as one billion guesses (6.8% to 0.5%).

Ignoring comprehensiveSubset passwords that were created under basic16 leaves 171 passwords, all created under less strict conditions than comprehensive8. Only 25.2% of these are cracked on average, suggesting that subsets drawn exclusively from less strict conditions are more difficult to guess than passwords created under stricter requirements.

To understand this result more deeply, we examined the

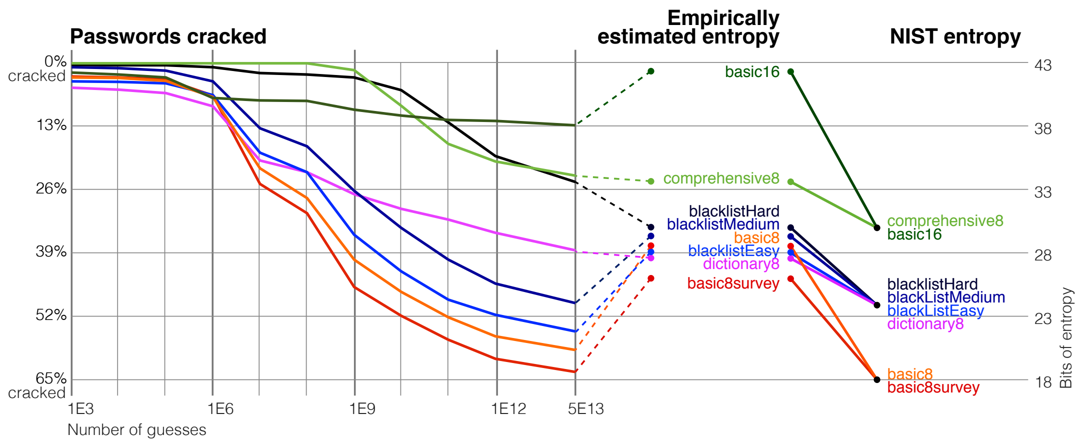

Figure 7. Relationship among the resistance of our collected password sets to heuristic cracking (experiment E); empirical entropy estimates we calculate from those sets; and NIST entropy estimates for our password conditions.

distribution of structures in the two test sets. There are 618 structures in the 1000-password comprehensive8 set, compared to 913 for comprehensiveSubset (normalized), indicating greater diversity in comprehensiveSubset passwords. This distribution of structures explains why comprehensive8 is significantly easier to guess.

We suspect this difference may be related to comprehensiveSubset isolating those users who make the most complex passwords. Regardless of the reason for this difference, however, researchers seeking to compare password policies should be aware that such subsets may not be representative.

D. *Guessability and entropy*

Historically, Shannon entropy (computed or estimated by various methods) has provided a convenient single statistic to summarize password strength. It remains unclear, however, how well entropy reflects the guess resistance of a password set. While information entropy does provide a theoretical lower bound on the guessability of a set of passwords [41], in practice a system administrator may be more concerned about how many passwords can be cracked in a given number of guesses than about the average guessability across the population. Although there is no mathematical relationship between entropy and this definition of guess resistance, we examine whether the two are correlated in practice. To do this, we consider two independent measures of entropy, as defined in Section IV-B: an empirically calculated estimate and a NIST estimate. For both measures, we find that entropy estimates roughly indicate which composition policies provide more guess resistance than others, but provide no useful information about the magnitude of these differences.

Empirically estimated entropy. We ranked our password conditions based on the proportion of passwords cracked in our most complete experiment (E) at one trillion guesses, and compared this to the rank of conditions based on empirically estimated entropy. We found these rankings, shown in Figure 7, to be significantly correlated (Kendall's $\tau = 0.71$, Holm-corrected $p = 0.042$). However, at one million or one billion guesses, the correlation in rankings is no longer significant (Holm-corrected $p = 0.275, 0.062$). We found the same pattern, correlation at one trillion guesses but not one billion or one million, in our largest public-data experiment (P4). These results indicate entropy might be useful when considering an adversary who can make a large number of guesses, but not when considering a smaller number of guesses.

Further, empirically estimated entropy did not predict the ranking of dictionary8, even when considering a large number of guesses. This condition displayed greater resistance to guessing than basic8, yet its empirically estimated entropy was lower. This might indicate a flaw in entropy estimation, a flaw in the guessing algorithm, or an innate shortcoming of the use of entropy to predict guessability. Since entropy can only lower-bound the guessability of passwords, it is possible for the frequency distribution of dictionary8 to have low entropy but high guess resistance. If this is the case, Verheul theorized that such a distribution would be optimal for password policy [51].

NIST entropy. Computing the NIST entropy of our password conditions produces three equivalence classes, as shown in Figure 7, because the heuristics are too coarse to capture all differences between our conditions. First, NIST entropy does not take into account the size of a dictionary or details of its implementation, such as case-insensitivity or removal of non-alphabetical characters before the check. All five of our dictionary and blacklist conditions meet the NIST requirement of a dictionary with at least 50,000 words [11]. Our results show that these variations lead to password policies with very different levels of password strength, which should be considered in a future heuristic.

Second, the NIST entropy scores for basic16 and comprehensive8 are the same, even though basic16 appears to be much more resistant to powerful guessing attacks. This may suggest that future heuristics should assign greater value to length than does the NIST heuristic.

Perhaps surprisingly, the equivalence classes given by NIST entropy are ordered correctly based on our results for guessability after 50 trillion guesses. Though it fails to capture fine-grained differences between similar password conditions, NIST entropy seems to succeed at its stated purpose of providing a "rough rule of thumb" [11].

We stress that although both measures of entropy provide a rough ordering among policies, they do not always correctly classify guessability (see for example dictionary8), and they do not effectively measure how much additional guess resistance one policy provides as compared to another. These results suggest that a "rough rule of thumb" may be the limit of entropy's usefulness as a metric.

VI. Discussion

We next discuss issues regarding ethics, ecological validity, and the limitations of our methodology.

Ethical considerations. Most of our results rely on passwords collected via a user study (approved by our institution's IRB). However, we also use the RockYou and MySpace password lists. Although these have collectively been used by a number of scientific works that study passwords (e.g., [2], [13], [25], [30]), this nevertheless creates an ethical conundrum: Should our research use passwords acquired illicitly? Since this data has already been made public and is easily available, using it in our research does not increase the harm to the victims. We use these passwords only to train and test guessing algorithms, and not in relationship with any usernames or other login information. Furthermore, as attackers are likely to use these password sets as training sets or cracking dictionaries, our use of them to evaluate password strength implies our results are more likely to be of practical relevance to security administrators.

Ecological validity. As with any user study, our results must be understood in context. As we describe in Section I, our participants are somewhat younger and more educated than the general population, but more diverse than typical small-sample password studies. The passwords we collected did not protect high-value accounts, reflecting a long-standing limitation of password research.

To further understand this context, we tested two password-creation scenarios (Section III-C): a survey scenario directly observing user behavior with a short-term, low-value account, and an email scenario simulating a longer-term, higher-value account. In both cases, users knew they might be asked to return and recall the password. Our users provided stronger passwords (measured by guessability and entropy) in the email scenario, a result consistent with users picking better passwords to protect a (hypothetical) high-value e-mail account than a low-value survey account.

To get real-world measures of password-related behavior, we surveyed users of Carnegie Mellon University's email system, which uses the comprehensive8 policy [9]. Comparing these survey results to the reports of our MTurk study participants, we find that on several measures of behavior and sentiment, the university responses (n = 280) are closer to those of our comprehensive8 participants than those of any other condition. For example, we asked MTurk participants who returned for the second half of the study whether they stored the password they had created (reassuring them they would get paid either way); we similarly asked university participants whether they store their login passwords. 59% of the university respondents report writing down their password, compared with 52% of comprehensive8 participants and a maximum of 37% for other MTurk conditions. These results show that study participants make different decisions based on password-composition requirements, and that in one condition their behavior is similar to people using that policy in practice.

We designed our study to minimize the impact of sampling and account-value limitations. All our findings result from comparisons *between* conditions. Behavior differences *caused by* the ways in which conditions differ (e.g., using a different technique to choose longer passwords than shorter ones) would be correctly captured and appropriately reflected in the results. Thus, we believe it likely that our findings hold in general, for at least some classes of passwords and users.

Other limitations. We tested all password sets with a number of password-guessing tools; the one we focus on (the Weir algorithm) always performed best. There may exist algorithms or training sets that would be more effective at guessing passwords than anything we tested. While this might affect some of our conclusions, we believe that most of them are robust, partly because many of our results are supported by multiple experiments and metrics.

In this work, we focused on automated offline password-guessing attacks. There are many other real-life threats to password security, such as phishing and shoulder surfing. Our analyses do not account for these. The password-composition policies we tested can induce different behaviors, e.g., writing down or forgetting passwords or using password managers, that affect password security. We report on some of these behaviors in prior work [46], but space constraints dictate that a comprehensive investigation is beyond the scope of this paper.

VII. Conclusion

Although the number and complexity of password-composition requirements imposed by systems administrators have been steadily increasing, the actual value added by these requirements is poorly understood. This work takes a substantial step forward in understanding not only these requirements, but also the process of evaluating them.

We introduced a new, efficient technique for evaluating password strength, which can be implemented for a variety of password-guessing algorithms and tuned using a variety of training sets to gain insight into the comparative guess resistance of different sets of passwords. Using this technique, we performed a more comprehensive password analysis than had previously been possible.

We found several notable results about the comparative strength of different composition policies. Although NIST considers basic16 and comprehensive8 equivalent, we found that basic16 is superior against large numbers of guesses. Combined with a prior result that basic16 is also easier for users [46], this suggests basic16 is the better policy choice. We also found that the effectiveness of a dictionary check depends heavily on the choice of dictionary; in particular, a large blacklist created using state-of-the-art password-guessing techniques is much more effective than a standard dictionary at preventing users from choosing easily guessed passwords.

Our results also reveal important information about conducting guess-resistance analysis. Effective attacks on passwords created under complex or rare-in-practice composition policies require access to abundant, closely matched training data. In addition, this type of password set cannot be characterized correctly simply by selecting a subset of conforming passwords from a larger corpus; such a subset is unlikely to be representative of passwords created under the policy in question. Finally, we report that Shannon entropy, though a convenient single-statistic metric of password strength, provides only a rough correlation with guess resistance and is unable to correctly predict quantitative differences in guessability among password sets.

VIII. Acknowledgments

We thank Mitch Franzos of the Carnegie Mellon Parallel Data Lab. We thank Stuart Schechter and Joe Bonneau for their feedback. This research was supported by NSF grants DGE-0903659, CNS-111676, and CCF-0424422; by CyLab at Carnegie Mellon under grant W911NF-09-1-0273 from the Army Research Office; and by Microsoft Research.

References

[1] D. Florêncio and C. Herley, "A large-scale study of web password habits," in *Proc. WWW'07*, 2007.

[2] M. Dell'Amico, P. Michiardi, and Y. Roudier, "Password strength: An empirical analysis," in *Proc. INFOCOM 2010*, 2010.

[3] M. Bishop and D. V. Klein, "Improving system security via proactive password checking," *Computers & Security*, vol. 14, no. 3, pp. 233–249, 1995.

[4] L. Constantin, "Sony stresses that PSN passwords were hashed," http://news.softpedia.com/news/Sony-Stresses-PSN-Passwords-Were-Hashed-198218.shtml, May 2011.

[5] P. Bright, "Anonymous speaks: The inside story of the HBGary hack," http://arst.ch/q6g, February 2011.

[6] J. Bonneau, "The Gawker hack: how a million passwords were lost," Dec. 2010, http://www.lightbluetouchpaper.org/2010/12/15/the-gawker-hack-how-a-million-passwords- were-lost/.

[7] P. Bright, "'Military meltdown Monday': 90k military usernames, hashes released," http://arstechnica.com/tech-policy/news/2011/07/military-meltdown-monday-90k-military-usernames-hashes-released.ars, 2011.

[8] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proc. SOUPS*, 2006.

[9] R. Shay, S. Komanduri, P. Kelley, P. Leon, M. Mazurek, L. Bauer, N. Christin, and L. Cranor, "Encountering stronger password requirements: user attitudes and behaviors," in *Proc. SOUPS '10*, 2010.

[10] L. St. Clair, L. Johansen, W. Enck, M. Pirretti, P. Traynor, P. McDaniel, and T. Jaeger, "Password exhaustion: Predicting the end of password usefulness," in *Proc. ICISS*, 2006.

[11] W. E. Burr, D. F. Dodson, and W. T. Polk, "Electronic authentication guideline," NIST, Tech. Rep., 2006.

[12] R. W. Proctor, M.-C. Lien, K.-P. L. Vu, E. E. Schultz, and G. Salvendy, "Improving computer security for authentication of users: Influence of proactive password restrictions," *Behavior Res. Methods, Instruments, & Computers*, vol. 34, no. 2, pp. 163–169, 2002.

[13] M. Weir, S. Aggarwal, M. Collins, and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in *Proc. CCS*, 2010.

[14] J. Ross, L. Irani, M. S. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers? Shifting demographics in Mechanical Turk," in *Proc. CHI EA*, 2010.

[15] P. G. Ipeirotis, "Demographics of Mechanical Turk," New York University, Tech. Rep. CeDER-10-01, 2010.

[16] M. Buhrmester, T. Kwang, and S. D. Gosling, "Amazon's Mechanical Turk: A new source of inexpensive, yet high-quality, data?" *Perspectives on Psychological Science*, vol. 6, no. 1, pp. 3–5, 2011.

[17] J. S. Downs, M. B. Holbrook, S. Sheng, and L. F. Cranor, "Are your participants gaming the system? Screening Mechanical Turk workers," in *Proc. CHI*, 2010.

[18] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with Mechanical Turk," in *Proc. CHI*, 2008.

[19] M. Toomim, T. Kriplean, C. Pörtner, and J. Landay, "Utility of human-computer interactions: toward a science of preference measurement," in *Proc. CHI*, 2011.

[20] S. Chiasson, A. Forget, E. Stobert, P. C. van Oorschot, and R. Biddle, "Multiple password interference in text passwords and click-based graphical passwords," in *Proc. CCS*, 2009.

[21] C. Kuo, S. Romanosky, and L. F. Cranor, "Human selection of mnemonic phrase-based passwords," in *Proc. SOUPS*, 2006.

[22] H. Wimberly and L. M. Liebrock, "Using fingerprint authentication to reduce system security: An empirical study," in *Proc. IEEE Symposium on Security and Privacy*, 2011.

[23] D. Davis, F. Monrose, and M. K. Reiter, "On user choice in graphical password schemes," in *Proc. USENIX Security Symposium*, 2004.

[24] A. Vance, "If your password is 123456, just make it hackme," New York Times, http://nyti.ms/w8NNwD, January 2010.

[25] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. IEEE Symposium on Security and Privacy*, 2009.

[26] D. Hart, "Attitudes and practices of students towards password security," *Journal of Computing Sciences in Colleges*, vol. 23, no. 5, pp. 169–174, 2008.

[27] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proc. CCS*, 2005.

[28] M. Zviran and W. J. Haga, "Password security: an empirical study," *J. Mgt. Info. Sys.*, vol. 15, no. 4, 1999.

[29] S. Komanduri and D. R. Hutchings, "Order and entropy in picture passwords," in *Graphics Interface*, 2008.

[30] J. Bonneau, M. Just, and G. Matthews, "What's in a name? Evaluating statistical attacks on personal knowledge questions," in *Proc. Financial Crypto. 2010*, 2010.

[31] Y. Zhang, F. Monrose, and M. K. Reiter, "The security of modern password expiration: an algorithmic framework and empirical analysis," in *Proc. CCS*, 2010.

[32] K.-P. L. Vu, R. W. Proctor, A. Bhargav-Spantzel, B.-L. B. Tai, and J. Cook, "Improving password security and memorability to protect personal and organizational information," *Int. J. of Human-Comp. Studies*, vol. 65, no. 8, pp. 744–757, 2007.

[33] A. Adams, M. A. Sasse, and P. Lunt, "Making passwords secure and usable," in *HCI 97*, 1997.

[34] P. Inglesant and M. A. Sasse, "The true cost of unusable password policies: password use in the wild," in *Proc. ACM CHI'10*, 2010, pp. 383–392.

[35] R. Shay, A. Bhargav-Spantzel, and E. Bertino, "Password policy simulation and analysis," in *ACM workshop on Digital identity management*, 2007, pp. 1–10.

[36] R. Shay and E. Bertino, "A comprehensive simulation tool for the analysis of password policies," *Int. J. Info. Sec.*, vol. 8, no. 4, pp. 275–289, 2009.

[37] J. M. Stanton, K. R. Stam, P. Mastrangelo, and J. Jolton, "Analysis of end user security behaviors," *Comp. & Security*, vol. 24, no. 2, pp. 124 – 133, 2005.

[38] D. Florêncio and C. Herley, "Where do security policies come from?" in *Proc. SOUPS '10*, 2010.

[39] S. Schechter, C. Herley, and M. Mitzenmacher, "Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks," in *Proc. HotSec'10*, 2010.

[40] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 1949.

[41] J. L. Massey, "Guessing and entropy," in *Proc. IEEE Int. Symp. Info. Theory*, 1994, p. 204.

[42] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive Password-Strength meters from markov models," in *Proc. NDSS 2012*, 2012.

[43] S. Marechal, "Advances in password cracking," *Journal in Computer Virology*, vol. 4, no. 1, pp. 73–81, 2008.

[44] C. M. Weir, "Using probabilistic techniques to aid in password cracking attacks," Ph.D. dissertation, 2010.

[45] S. Designer, "John the Ripper," http://www.openwall.com/john/, 1996-2010.

[46] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of passwords and people: Measuring the effect of password-composition policies," in *Proc. CHI*, 2011.

[47] B. Schneier, "Myspace passwords aren't so dumb," http://www.wired.com/politics/security/commentary/securitymatters/2006/12/72300, December 2006.

[48] T. White, *Hadoop: The Definitive Guide*, 2nd ed. O'Reilly, September 2010.

[49] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. OSDI*, 2004.

[50] C. E. Shannon, "Prediction and entropy of printed english," *Bell Systems Tech. J.*, vol. 30, 1951.

[51] E. Verheul, "Selecting secure passwords," in *Topics in Cryptology–CT-RSA 2007*, 2006.

Appendix

Here we detail the complete training and test data used in each of our Weir-algorithm experiments. The first column gives the experiment number. The next three columns list the three types of training data used to create a Weir-calculator experiment. The *structures* column shows the wordlists used to generate the set of character-type structures that define the Weir algorithm's search space. The wordlists in the *digits and symbols* column determine the probabilities for filling combinations of digits and symbols into these structures. The wordlists in the *strings* column determine the probabilities for filling alphabetic strings into structures. In most cases, we train strings on as much data as possible, while restricting structure and digit/symbol training to wordlists that contain a quality sample of multi-character-class passwords. The final column describes the password set(s) we attempted to guess.

We also list the complete training and test data used in each of our BFM experiments. The experiment number and test set columns are the same as in the Weir subtable. Training for the BFM calculator, however, uses only one combined wordlist per experiment; these lists are detailed in the *training set* column.

Abbreviations for all the training and test sets we use are defined in the key below the tables.

Weir experiment descriptions

Name	**Training sets**			**Testing Set**
	Structures	Digits and symbols	Strings	
Trained from public password data				
P1	MS8	MS	MS	1000-All
P2	MS8	MS	MS, W2, I	1000-All
P3	MS8	MS, RY	MS, W2, I, RY	1000-All
P3-C8	MSC	MS, RY	MS, W2, I, RY	1000-C8
P3-B16	MS16	MS, RY	MS, W2, I, RY	1000-B16
P4	MS8, OW8	MS, RY, OW	MS, W2, I, RY, OW	1000-All
P4-B16	MS16, OW16	MS, RY, OW	MS, W2, I, RY, OW	1000-B16
Trained on half of our dataset, weighted to 1/10th, equal-size, or 10x the cumulative size of the public data				
X1/10	MS8, 500-All	MS, RY, 500-All	MS, W2, I, RY, 500-All	500-All
X1	MS8, 500-All	MS, RY, 500-All	MS, W2, I, RY, 500-All	500-All
X10	MS8, 500-All	MS, RY, 500-All	MS, W2, I, RY, 500-All	500-All
Everything				
E	MS8, OW8, 500-All	MS, RY, OW, 500-All	MS, W2, I, RY, OW, 500-All	500-All
Testing password subsets that meet comprehensive8 requirements				
S0a	MSC, OWC	MS, OW	MS, W2, I, OW	1000-C8, 206-C8S
S0b	MSC, OWC, 2000-C8	MS, OW, 2000-C8	MS, W2, I, OW, 2000-C8	1000-C8, 206-C8S
S1	MSC, OWC, RYCD	MS, OW, RY	MS, W2, I, OW, RY	1000-C8, 206-C8S
S2	MSC, OWC, 2000-C8, RYCD	MS, OW, 2000-C8, RY	MS, W2, I, OW, 2000C8, RY	1000-C8, 206-C8S
Split ratio testing on basic8				
B8a	MS8, OW8, 500-B8	MS, RY, OW, 500-B8	MS, W2, I, RY, OW, 500-B8	2500-B8
B8b	MS8, OW8, 1000-B8	MS, RY, OW, 1000-B8	MS, W2, I, RY, OW, 1000-B8	2000-B8
B8c	MS8, OW8, 1500-B8	MS, RY, OW, 1500-B8	MS, W2, I, RY, OW, 1500-B8	1500-B8
B8d	MS8, OW8, 2000-B8	MS, RY, OW, 2000-B8	MS, W2, I, RY, OW, 2000-B8	1000-B8
B8e	MS8, OW8, 2500-B8	MS, RY, OW, 2500-B8	MS, W2, I, RY, OW, 2500-B8	500-B8
Split ratio testing on comprehensive8				
C8test1/10	MSC, 500-C8	MS, RY, 500-C8	MS, W2, I, RY, 500-C8	2500-C8
C8test1	MSC, 500-C8	MS, RY, 500-C8	MS, W2, I, RY, 500-C8	2500-C8
C8a	MSC, OWC, 500-C8	MS, RY, OW, 500-C8	MS, W2, I, RY, OW, 500-C8	2500-C8
C8b	MSC, OWC, 1000-C8	MS, RY, OW, 1000-C8	MS, W2, I, RY, OW, 1000-C8	2000-C8
C8c	MSC, OWC, 1500-C8	MS, RY, OW, 1500-C8	MS, W2, I, RY, OW, 1500-C8	1500-C8
C8d	MSC, OWC, 2000-C8	MS, RY, OW, 2000-C8	MS, W2, I, RY, OW, 2000-C8	1000-C8
C8e	MSC, OWC, 2500-C8	MS, RY, OW, 2500-C8	MS, W2, I, RY, OW, 2500-C8	500-C8

BFM experiment descriptions

Name	Training set	Test set
B1	RY, MS, I	1000-All
B2	RY, MS, I, 500-All	500-All
B3	RY, MS, I, 2000-B8	1000-B8
B4	RY, MS, I, 2000-C8	1000-C8

Key to password sets

RY	RockYou list	**I**	inflection list
RYCD	RY, filtered w/ all reqs. of C8	**W2**	simple Unix dictionary
MS	MySpace list	**OW**	paid Openwall dictionary
MS8	MS, filtered w/ min length of 8	**OW8**	OW, filtered w/ min length of 8
MS16	MS, filtered w/ min length of 16	**OW16**	OW, filtered w/ min length of 16
MSC	MS, filtered w/ min length of 8 and character class reqs. of C8	**OWC**	OW, filtered w/ min length 8 and character class reqs. of C8
n**-All**	n passwords from each of our conditions	n**-B8**	n basic8 passwords
n**-B16**	n basic16 passwords	n**-C8**	n comprehensive8 passwords
n**-C8S**	n comprehensiveSubset passwords	n**-RYCD**	n RYCD passwords