Improving the Eco-system of Passwords

Weining Yang
Purdue University

Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Weining Yang

Entitled
Improving the Eco-System of Passwords

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Ninghui Li	
Chair	
Dan Goldwasser	
Robert W. Proctor	
Elisa Bertino	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Ninghui Li

Approved by: William J. Gorman 06/24/2016

Head of the Departmental Graduate Program Date

IMPROVING THE ECO-SYSTEM OF PASSWORDS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Weining Yang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2016

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

My deepest gratitude goes to my advisor Dr. Ninghui Li, for his invaluable guidance and his countless suggestions which cultivating me into an independent researcher. I am fortunate to have the opportunity to interact with such a productive and respected mentor who also understood the need for balance. This dissertation would not have been materialized without his help.

My appreciation also goes to my Ph.D. committee members, Dr. Elisa Bertino, Dr. Dan Goldwasser, and Dr. Robert W. Proctor for their helpful advice and suggestions during my preliminary exam and on my dissertation. I would like to extend my thanks to Dr. William J. Gorman for his efforts on helping format my dissertation.

Special thanks go to my lab mates, Wahbeh Qardaji, Christopher Gates, Dong Su, Haining Chen, Wei-Yen Day, Sze Yiu Chau, Huangyi Ge, and Tianhao Wang, for their valuable collaborations and assistance on my research, as well as for the great friendship.

Finally, I would like to thank my family for their encouragement, understanding, and limitless support during my Ph.D. studies.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES ix

ABSTRACT xi

1 INTRODUCTION 1

2 BACKGROUND AND RELATED WORK 7

2.1 Password Strength Evaluation Metrics 7

2.2 Evaluating User-chosen Passwords and Password Policies 12

2.3 Password Generation Strategies 14

3 IMPROVING METHODOLOGY TO HELP PASSWORD RESEARCH . . . 18

3.1 Introduction 18

3.2 Password Models and Metrics 22

3.3 Design Space of Password Models 26

3.3.1 Whole-String Markov Models 27

3.3.2 Template-based Models 33

3.3.3 Password Generation 35

3.4 Experimental Methodologies 37

3.5 Experimental Results 46

3.6 Conclusions 57

4 IMPROVING PASSWORD POLICIES AND PRACTICE USED BY WEBSITES 59

4.1 Introduction 59

4.2 How to Compare PRAs 62

4.2.1 Guess Number Graph (GNG) 62

4.2.2 The β-Residual Strength Graph (β-RSG) 63

4.2.3 The Normalized β-Residual Strength Graph (β-NRSG) 64

Page

4.2.4 Client versus Server PRAs 65

4.2.5 PRAs We Consider 65

4.3 Data Cleansing 70

4.4 Experimental Results 75

4.4.1 Experimental Datasets and Settings 75

4.4.2 Experimental Results 77

4.5 Conclusion 83

5 UNDERSTANDING MNEMONIC SENTENCE-BASED PASSWORD GENERATION STRATEGIES 85

5.1 Introduction 85

5.2 Study 1: Security 88

5.2.1 Study Design 93

5.2.2 Methodology 94

5.3 Results From Study 1 95

5.3.1 Analyzing Passwords Using Probability Models and Password Strength Meters. 96

5.3.2 Strength of Mnemonic Sentences 98

5.3.3 Cracking Mnemonic Passwords 106

5.4 Study 2: Usability 111

5.4.1 Study Overview 112

5.4.2 First Phase Results 113

5.4.3 Second Phase Results 116

5.5 Discussion and Conclusion 118

6 UNDERSTANDING WORD-BASED PASSWORD GENERATION STRATEGIES 120

6.1 Introduction 120

6.2 Study 1: Security 121

6.2.1 Study Design 124

6.3 Results From Study 1 125

Page

6.3.1 Analyzing Passwords Using Probability Models and Password Strength Meters. 125

6.3.2 Analyzing Passwords Using Statistical Quantities. 127

6.4 Study 2: Usability 134

6.4.1 First Phase Results 135

6.4.2 Second Phase Results 138

6.5 Conclusion 139

7 SUMMARY 141

REFERENCES 144

VITA 149

LIST OF TABLES

Table — Page

3.1 Design space for probabilistic password models 35

3.2 Results of removing passwords that are too long, too short, and non-ASCII passwords. 38

3.3 Password count and frequency information of all datasets 39

3.4 Password count and frequency information of American datasets 39

3.5 Password count and frequency information of Chinese datasets 40

3.6 Password length information for English datasets 40

3.7 Password length information for Chinese password datasets 41

3.8 Password characters information 43

3.9 Password pattern information: **L** denotes a lower-case sequence, **D** denotes a digit sequence, **U** denotes a upper-case sequence, and **S** denotes a symbol sequence; patterns differ from templates in that patterns do not record length of sequence. 44

3.10 Percentage of password patterns in datasets: **L** denotes a lower-case sequence, **D** denotes a digit sequence, **U** denotes a upper-case sequence, and **S** denotes a symbol sequence; patterns differ from templates in that patterns do not record length of sequence 45

3.11 Six experimental scenarios. 46

3.12 $\text{ANLL}_{0.8}$ results for scenarios 1, 2, 3. end, dir, and dis stand for end-symbol, direct, and distribution-based normalization, respectively. We highlight the best results within each scenario. 52

3.13 $\text{ANLL}_{0.8}$ results for scenarios 4, 5, 6. end, dir, and dis stand for end-symbol, direct, and distribution-based normalization, respectively. We highlight the best results within each scenario. 54

4.1 Server-end PRAs and Client-end PRAs. X_c means reduced-size version of model X in order to be deployed at the client side. 66

4.2 Model size of Markov models with backoff using different frequency threshold. 67

4.3 Examples of differences in two versions of *zxcvbn*. 70

Table Page

4.4 Examples of problematic IDs in *178* dataset. 71

4.5 Number of accounts removed after identifying problematic IDs. 72

4.6 Top 10 passwords with most accounts removed from English datasets. $pwd_{r/o}$ means the original count of pwd in the dataset is o, and r accounts are removed. 74

4.7 Top 10 passwords with most accounts removed from Chinese datasets. $pwd_{r/o}$ means the original count of pwd in the dataset is o, and r accounts are removed. 74

4.8 Top 20 words in the new dictionary for Chinese passwords used in *zxcvbn*. . 77

4.9 y values of GNG and β-RSG when $x = 10^4$ and $x = 10^6$ evaluated on English datasets. *Y+P* stands for *Yahoo* + *Phpbb*. $\beta = 10$ 82

4.10 y values of GNG and β-RSG when $x = 10^4$ and $x = 10^6$ evaluated on Chinese datasets. $\beta = 10$ 82

5.1 Mnemonic-based strategy description 88

5.2 Average length of passwords in each strategy as well as Yahoo and Phpbb datasets. 97

5.3 $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) in Control as well as samples with size 800 from Rockyou, Phpbb, and Yahoo. E_{SD} means the average is E and the standard deviation is SD. 98

5.4 $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) in mnemonic strategies. 99

5.5 Popular passwords and probability for top 5 frequently chosen sentences in mnemonic strategies. 100

5.6 Potential popular and personalized sentence counts evaluated by Google search results. 105

5.7 Character usage in mnemonic strategies. 108

5.8 Statistics for first-phase usability study for mnemonic strategies. 114

5.9 Statistics for second-phase usability study for mnemonic strategies. 115

6.1 Word-based strategy description 122

6.2 Word-based strategies used in each round of the study. 124

6.3 $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) in word-based strategies. 128

6.5 Popular passwords and probability for top 5 frequently chosen word combinations in word-based strategies. 129

6.4 Word frequency in word-based strategies. 132

Table Page

6.6 Punctuation frequency in 2WordPunEx and 2WordPun. 133

6.7 Statistics for first-phase usability study for word-based strategies. 135

6.8 Statistics for second-phase usability study for word-based strategies. 136

LIST OF FIGURES

Figure Page

3.1 An example of probability-threshold graph and guess-number graph 24

3.2 Guess number graphs and probability threshold graphs for Scenario 1: Rock$\rightarrow$Ya+Ph. (Part 1) 47

3.3 Guess number graphs and probability threshold graphs for Scenario 1: Rock$\rightarrow$Ya+Ph. (Part 2) 48

3.4 Guess number graphs and probability threshold graphs for Scenario 2: Du+178$\rightarrow$CSDN. (Part 1) 50

3.5 Guess number graphs and probability threshold graphs for Scenario 2: Du+178$\rightarrow$CSDN. (Part 2) 51

3.6 Guess number graphs and probability threshold graphs for Scenarios 3, 4. . 52

3.7 Guess number graphs and probability threshold graphs for Scenarios 5, 6. . 54

4.1 β-Residual Strength Graph(β-RSG) on original *Xato* and *178* datasets. A point (x, y) on a curve means if we want to reject the top x passwords from a PRA, the strength of the remaining passwords is y. 71

4.2 CDF of $-\log_{10} p$ for all IDs with length 10 in all datasets. p is the probability of IDs. The dashed lines are CDF of normal distribution with the same mean and standard deviation 73

4.3 The Guess Number Graph (GNG), the β-Residual Strength Graph (β-RSG), and the Normalized β-Residual Strength Graph (β-NRSG) evaluated on *Xato* dataset. 78

4.4 The Guess Number Graph (GNG), the β-Residual Strength Graph (β-RSG), and the Normalized β-Residual Strength Graph (β-NRSG) evaluated on *178* dataset. 79

5.1 Comparison of strength of passwords resulted from different mnemonic strategies using probabilistic models and password strength meters. 96

5.2 Percentage of passwords cracked within 10 attempts for case-insensitive passwords, and 20 attempts for case-sensitive passwords. 109

5.3 Guess number graph on passwords created by using mnemonic strategies. . 110

Figure Page

5.4 Mean scores of TLX as a function of strategy and subscale for Control, MneGenEx, and MnePerEx. Error bars represent standard errors of the scores. 116

6.1 Comparison of strength of passwords from different word-based strategies and datasets using probabilistic models and password strength meters. 126

6.2 Mean scores of TLX as a function of strategy and subscale for Control, 3WordEx, and 4WordEx. Error bars represent standard errors of the scores. . 137

ABSTRACT

Yang, Weining PhD, Purdue University, August 2016. Improving the Eco-system of Passwords. Major Professor: Ninghui Li.

Password-based authentication is perhaps the most widely used method for user authentication. Passwords are both easy to understand and use, and easy to implement. With these advantages, password-based authentication is likely to stay as an important part of security in the foreseeable future. One major weakness of password-based authentication is that many users tend to choose weak passwords that are easy to guess. In this dissertation, we address the challenge and improve the eco-system of passwords in multiple aspects. Firstly, we provide methodologies that help password research. To be more specific, we propose Probability Threshold Graphs, which is superior to Guess Number Graphs when comparing password models and password datasets. We also introduce rich literature of statistical language modeling into password modeling and show that if used correctly, whole-string Markov models outperform Probabilistic Context Free Grammar models. Secondly, we improve password policies and practice used by websites by studying how to best check weak passwords. We model different password strength checking methods as Password Ranking Algorithms (PRAs), and introduce two methods for comparing different PRAs: the β-Residual Strength Graph and the Normalized β-Residual Strength Graph. Finally, we examine the security and usability of commonly suggested password generation strategies. We find that for mnemonic sentence-based strategies, differences in the exact instructions have a tremendous impact on the security level of the resulting passwords. For word-based strategies, security of the resulting passwords mainly depends on the number of words required, and requiring at least 3 words is likely to result in passwords stronger than the general passwords chosen by typical users.

1. INTRODUCTION

Password-based authentication is perhaps the most widely used method for user authentication. Passwords are both easy to understand and use, and easy to implement. With these advantages, password-based authentication is likely to stay as an important part of security in the foreseeable future [1]. Considering the large number of systems that require passwords and the fact that some systems mandate password changes, it is apparent that many users will face hundreds or more password-creation situations over a lifetime.

It is well known that there is an inherent tension between the security and usability of passwords [2, 3]. More precisely, secure passwords tend to be difficult to memorize (i.e., less usable) whereas passwords that are memorable tend to be predictable. Generally individuals side with usability of passwords by choosing predictable and weak passwords [2, 4–7]. Users tend to choose weak passwords that are easy to guess is a major weakness of password-based authentication. Addressing this challenge has been an active and important research area in recent years.

In this research, we improve the eco-system of passwords in multiple aspects.

Providing Methodology to Help Password Research Researchers have studied the quality of users' password choices under different scenarios [8–13]. In this area, earlier work uses either standard password cracking tools such as John the Ripper (JTR) [14], or ad hoc approaches for estimating the information entropy of a set of passwords. One such approach is NIST's recommended scheme [15] for estimating the entropy of one password. The entropy estimation is mainly based on passwords' length. Weir et al [16] argued that entropy estimation methods are inaccurate metrics for password strength. Instead, they suggest measuring the strength of user-chosen passwords by password models and rejecting passwords with high probabilities.

Later, probabilistic model-based methods and guess numbers became the standard method to measure the strength of passwords. Probabilistic models of passwords work by assigning a probability to each string. Some models divide a password into several segments, often by grouping consecutive characters of the same category (e.g., lower-case letters, digits, etc.) into one segment, and then generate the probability for each segment independently. Examples include the model in [17], and the Probabilistic Context Free Grammar (PCFG)-based approach developed in [18]. A whole-string model, on the other hand, does not divide a password into segments, e.g., the Markov models in [19, 20].

The guess number of a password according to a password model is defined to be the rank of the password in the order of decreasing probability. To compare two sets of passwords, one plots the number of guesses vs. the percentage of passwords cracked by the corresponding number of guesses in the testing dataset. Such *guess-number graphs* are currently the standard tool in password research. Plotting such graphs, however, requires the computation of guess numbers of passwords, which is computationally expensive. For many password models, this requires generating all passwords with probability above a certain threshold and sorting them.

We find the current state of art unsatisfying. First, Markov models are known as n-gram models in the statistical language literature, and there exist a large number of techniques developed to improve the performance of such models. However, Password modeling research has not taken advantage of such knowledge and techniques. Second, passwords differ from statistical language modeling in that passwords have a very definite length distributions: most passwords are between lengths 6 and 12, and there are very few short and long passwords. Therefore, assuming uniform length distribution, which is the assumption in n-gram models is suboptimal. Finally, different password modeling approaches have not been systematically evaluated or compared against each other.

In Chapter 3, we propose probability-threshold graphs, which have important advantages over guess-number graphs. They are much faster to compute, and at the same time provide information beyond what is feasible in guess-number graphs. Based on that, we conduct an extensive empirical study of different password models using 6 real-world plaintext password datasets totaling about 60 million passwords, including 3 from American websites and 3 from Chinese websites. We consider three different normalization approaches: *direct*, which assumes that strings of all lengths are equally likely, *distribution-based*, which uses the length distribution in the training dataset, and *end-symbol based*, which appends an "end" symbol to each password for training and testing. We consider Markov chain models of different orders, and with different smoothing techniques. We compare whole-string models with different instantiation of template-based models, and find that Markov models, when done correctly, perform significantly better than the Probabilistic Context-Free Grammar model proposed in Weir et al. [18], which has been used as the state-of-the-art password model in recent research.

Improving Password Policies and Practice Used by Websites To deal with weak and predictable passwords chosen by users, the most common approach is to forbid the use of weak passwords, or give warnings for passwords that are "somewhat weak". This approach requires an effective way to identify weak passwords.

How to best check weak passwords is still an open question. A study in 2014 [21] examined several password meters in use at popular websites and found highly inconsistent strength estimates for the same passwords using different meters. The report did not answer the question of which meter is the best, nor what methods should be used to compare them. Designing an effective password meter requires solving two problems: (1) How to accurately assess the strength of passwords chosen by the users; and (2) How to communicate the strength information to and interact with the users to encourage them to choose strong passwords. These two problems are largely orthogonal. In this research we focus on solving the first problem.

In Chapter 4, we model different password strength assessing methods (including composition policies) as Password Ranking Algorithms (PRAs), which assign a rank to every password. One state-of-the-art method for comparing PRAs is the *guess-number graphs* (GNG), which plots the number of guesses vs. the percentage of passwords cracked in the test datasets. However, GNG measures only the total density of the uncracked passwords, but not their distribution, which is critical in assessing the effectiveness to defend against guessing attacks after deploying the PRA. To address this limitation of the GNG, we propose the β-Residual Strength Graph (β-RSG), which measures the strength of the β most common passwords in the test dataset, after forbidding the weakest passwords identified by a PRA. When a PRA forbids a large number of passwords that users are extremely unlikely to use, it performs poorly under β-RSG. To limit the influence of these passwords, we also propose Normalized β-Residual Strength Graph (β-NRSG), which ignores how passwords that do not appear in the testing dataset are ranked. β-NRSG also has the advantage that we can use it to evaluate blackbox password strength services for which one can query the strength of specific passwords, but cannot obtain all weak passwords.

We compare the Probabilistic Context-Free Grammar (PCFG) [18] method, Markov chain models with and without backoff [17, 19], blacklists based on training datasets, the Combined method proposed by Ur et al. [22], password composition policies, as well as two versions of *zxcvbn* [23]. We also show how GNG, β-RSG, and β-NRSG differ. We find that when one places no limit on the mode size, several methods including the blacklist approach, Markov M=models, and the Combined method have similar performance.

When one wants to check the strength of passwords on the client side, without sending passwords over the network, the model size must be limited. We find that a blacklist with a limited size still provide the most accurate strength measurement for the most popular passwords. However, only a limited number of passwords are covered. We then propose a new client-end PRA that uses a hybrid method; it uses a small blacklist to assess the strength of most popular passwords, and evaluates the other passwords based on a limited size Markov model with backoff. We show that the hybrid method inherits the advantages of both methods, and consistently outperforms the other client-end PRAs.

Improve Understanding Password Generation Strategy Another solution for avoiding weak passwords is to encourage users to choose strong passwords by educating users with password creation strategies.

The security community has been trying to come up with password generation strategies that can help users generate secure and usable passwords. Candidate strategies have been suggested by sources ranging from the National Institute of Standards and Technology (NIST) [24] to online comics [25], and from security experts' essays [26, 27] to online help forums. However, these suggestions are often based on intuitions instead of scientific knowledge. Little is actually known about which strategies are effective in helping users create usable and secure passwords.

In Chapter 5, we study the mnemonic sentence-based strategy (for short, the *mnemonic strategy*), which is the most widely recommended and studied strategy. It appears that the general assessment is that this is a good strategy. It is recommended by NIST [24] and by security experts [26, 27].

We conduct a large study to evaluate 6 variants of the mnemonic strategy and compare against a control group. When assessing the security of the strategies, we go beyond the methods used in existing studies in two ways. First, we adopt the approach of using statistical quantities to measure the distributions of the passwords, as articulated by Bonneau [28]. In particular, we chose to use the *β-guess-rate* (λ_β) [29], which measures the expected success for an attacker limited to β guesses per account. We chose to use $\beta = 1$ and $\beta = 10$, both because they were suggested in [30] as appropriate for defense against online guessing attacks and because a larger β is not very meaningful for our sample sizes. Second, we develop a method specific for attacking passwords resulted from the mnemonic strategy, and demonstrate the effectiveness of this attack.

The usability of the variants is evaluated in a separate user study, in which password creation time, short-term and long-term password memorability, and the workload required in both password creation and retention are evaluated.

We find that in terms of security, while metrics similar to guess numbers suggest that all variants provide highly secure passwords, statistical metrics tell a far more interesting and nuanced story. In particular, differences in the exact instructions have a tremendous impact on the security and usability of the resulted passwords, and a tradeoff between security and long-term memorability of the passwords is observed.

In Chapter 6, we evaluate variants of word-based strategies, which are also recommended by the security community [24, 25, 31]. In word-based strategies, a password is generally created in a two-step process: first choose some unrelated or random words (usually 2 to 4), then combine the words into a password. Despite the popularity of the strategies, to the best of our knowledge, the strategies have not been evaluated in any human subject studies and little is known about the effectiveness of them in helping users create usable and secure passwords.

We examine the security and usability of 7 variants of the strategy in a series of studies conducted on Mturk. The evaluation metrics are identical to that used in Chapter 5.

We find that the security of the variants depends on the number of words required. While the variants requiring two words are likely to produce passwords with similar security level to the baseline, strategies based on 3 or 4 words significantly reduce the number of word combinations that are chosen more than once, suggesting significantly stronger resulted passwords, where the variant requiring 4 words performs better. In terms of usability, 3-word based strategies and 4-word based strategies have similar performance in long-term password recall, and both of them underperfrom the baseline. We also observe that requiring the fourth word results in more difficulty in password creation.

2. BACKGROUND AND RELATED WORK

In this chapter, we briefly review background and related work on password strength evaluation metrics, evaluations on passwords and password policies, and password generation strategies.

2.1 Password Strength Evaluation Metrics

The quality of passwords has traditionally been measured using a combination of standard password cracking tools, such as John the Ripper (JTR) [14], and ad hoc approaches for estimating the information entropy of a set of passwords.

In 1990, Klein et al. [4] proposed the concept of a proactive password checker, which checks the strength of newly created passwords and prevents users from choosing weak passwords. Since then, multiple blacklist-based proactive password checkers were proposed. Spafford et al. [32] and Bergadano et al. [33] developed methods for filtering passwords based on efficiently stored password dictionaries. Manber et al. [34] described an approach that refused not only exact words in dictionaries but also passwords that are a single insertion, deletion, or substitution from a dictionary word. Yan et al. [35] suggested that besides dictionary checking, password checker should consider length and character types of passwords as well.

Bonneau [28] criticized the comparability and repeatability of past password cracking results using tools such as John the ripper and/or dictionaries. He proposed metrics for studying the overall level of security in large password datasets, based only on the distribution, and not on the actual password strings.

In terms of entropy estimation, Florencio and Herley [6], Forget et al. [36], and Egelman et al. [37] used the formula where the bit strength of a password is considered to be $\log_2\left(|\Sigma|^{len}\right)$ for alphabet Σ; for example, a 9-character password that contains both upper and lower case characters and digits is considered to have bit strength of $\log_2(26 + 26 + 10)^9 \approx 53.59$. A more sophisticated approach, known as the NIST guidelines [15], calculates password entropy using several factors, including how many numbers, symbols, and uppercase letters are used and where they appear. The NIST guidelines work by estimating the strength of password creation policies using Shannons entropy. In particular, for a password creation policy with a required length, it assigns entropy to characters at each position and the total entropy score is the sum of them with some additional bonus entropy if a composition rule is required or a dictionary check is performed. The entropy score for each character is assigned using the following criteria:

1. The entropy of the first character is taken to be 4 bits;

2. The entropy of the next 7 characters is 2 bits per character; this is roughly consistent with Shannons estimation that "when statistical effects extending over not more than 8 letters are considered the entropy is roughly 2.3 bits per character";

3. For the 9th through the 20th character the entropy is taken to be 1.5 bits per character;

4. For characters 21 and above the entropy is taken to be 1 bit per character;

5. A "bonus" of 6 bits of entropy is assigned for a composition rule that requires both upper case and non-alphabetic characters. This forces the use of these characters, but in many cases these characters will occur only at the beginning or the end of the passwords, and it reduces the total search space somewhat, so the benefit is probably modest and nearly independent of the length of the password;

6. A bonus of up to 6 bits of entropy is added for an extensive dictionary check. If the attacker knows the dictionary, he can avoid testing those passwords, and will in any event, be able to guess much of the dictionary, which will, however, be the most likely selected passwords in the absence of a dictionary rule. The assumption is that most of the guessing entropy benefits for a dictionary test accrue to relatively short passwords, because any long password that can be remembered must necessarily be

a pass-phrase composed of dictionary words, so the bonus declines to zero at 20 characters.

In 2010, Weir et al. [16] measured the strength of password creation policies by using large-scale real-world datasets and showed that entropy values would not tell the defender anything about how vulnerable a policy would be to an online password cracking attack. They also showed that many rule-based policies performed poorly for ensuring a desirable level of security. Instead, they suggested measuring the strength of user-chosen passwords by password models and rejecting passwords with high probabilities. Schechter et al. [38] also suggested allowing users to choose any password they want, so long as it was not already too popular with other users.

Later, probabilistic model-based methods became the standard method to measure the strength of passwords. Probabilistic models of passwords work by assigning a probability to each string. Some models divide a password into several segments, often by grouping consecutive characters of the same category (e.g., lower-case letters, digits, etc.) into one segment, and then generate the probability for each segment independently. Examples include the model in [17], and the Probabilistic Context Free Grammar (PCFG)-based approach developed in [18]. A whole-string model, on the other hand, does not divide a password into segments, e.g., the Markov chain model in [19, 20].

Narayanan and Shmatikov [17] proposed a template-based model, with Markov chain being used for assigning probability to letter-based segments.

The PCFG model was proposed by Weir et al. [18]. In the approach, one divides a password into several segments by grouping consecutive characters of the same category (e.g., letters, digits, special symbols) into one segment. Each password thus follows a pattern, for example, L_7D_3 denotes a password consisting of a sequence of 7 letters followed by 3 digits. The distribution of different patterns as well as the distribution of digits and symbols are learned from a training dataset. PCFG chooses words to instantiate segments consisting of letters from a dictionary where all words in the dictionary are assumed to be of the same probability. The probability of a password is calculated by multiplying the probability of the pattern by the probabilities of the particular ways the segments are instantiated. The PCFG approach was later improved by introducing semantic patterns [39], or by using specialized data in training [8].

N-gram models, i.e., Markov chains, have been applied to passwords [19]. A Markov chain of order d, where d is a positive integer, is a process that satisfies

$$P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) = P(x_i|x_{i-1}, \ldots, x_{i-d})$$

where d is finite and $x_1, x_2, x_3, \ldots$ is a sequence of random variables. A Markov chain with order d corresponds to an n-gram model with $n = d + 1$. For example, a 2-order Markov chain, in which the probability assigned to a password "$c_1c_2 \cdots c_\ell$" is

$$P(\text{“}c_1c_2 \cdots c_\ell\text{”}) = P(c_1|c_0)P(c_2|c_1)P(c_3|c_2) \cdots P(c_\ell|c_{\ell-1}),$$

where $c_0 \notin \Sigma$ denotes a start symbol that is prepended to the beginning of all passwords, and

$$P(c_i|c_{i-1}) = \frac{count(c_{i-1}c_i)}{count(c_{i-1}\cdot)} \tag{2.1}$$

where $count(c_{i-1}\cdot)$ denotes the number of occurrences of c_{i-1} where it is followed by another character (i.e., where it is not at the end of password), and $count(c_{i-1}c_i)$ gives the number of occurrences of the substring $c_{i-1}c_i$. By definition, we have $count(c_{i-1}\cdot) = \sum_{c_i \in \Sigma} count(c_{i-1}c_i)$.

Dell'Amico et al. [40] compared the cracking effectiveness of Markov models on the "Italian", "Finnish", and "MySpace" datasets by computing the estimated probabilities for each password in the datasets and using an approximation algorithm to compute the number of guesses for each threshold. They did not consider normalization, smoothing, etc. Furthermore, their results showed very small search spaces for higher-order Markov models, which seemed to suggest that their approximation algorithm underestimated the search space.

Other commonly used password strength measurement tools include John the Ripper [14] and *zxcvbn* [23].

John the Ripper [14], one of the most popular password cracking tools, has several modes for generating password guesses. The wordlist mode uses a dictionary plus various mangling rules. The incremental mode uses trigraph frequencies, i.e., frequencies for the triple of character, position, and password length. The Markov mode uses Markov chains of order 1.

Zxcvbn is an open-source meter designed to run entirely in clients' browser developed by Wheeler [23]. It decomposes a given password into patterns, and then assigns each pattern an estimated "entropy". The entropy of each chunk is estimated depending on the pattern of the chunk. The candidate patterns are "dictionary", "sequence", "spatial", "year", "date", "repeat" and "bruteforce". For example, if a chunk is within the pattern "dictionary", the entropy is estimated as the log of the rank of word in the dictionary. Additional entropy is added if uppercase letters are used or some letters are converted into digits or sequences (e.g. a$\Rightarrow$@). There are 5 embedded frequency-ordered dictionaries: 7140 passwords from the Top 10000 password dictionary; and three dictionaries for common names from the 2000 US Census. After chunking, a password's entropy is calculated as the sum of its constituent chunks' entropy estimates.

$$\mathsf{entropy}(pwd) = \sum \mathsf{entropy}(chunk_i)$$

A password may be divided into chunks in different ways, *Zxcvbn* finds the way that yields the minimal entropy and uses that.

In October 2015, a new version of *zxcvbn* was published. The new version of *zxcvbn* also divides a password into chunks, and computes a password's strength as the "minimal guess" of it under any way of dividing it into chunks. A password's "guesses" after being divided into chunks under a specific way is:

$$l! \times \prod_{i=1}^{l} (\mathit{chunk}_i.\mathit{guesses}) + 10000^{l-1}$$

where l is the number of the chunks. The factorial term is the number of ways to order l patterns. The $10000^{(l-1)}$ term captures the intuition that a password that has more chunks are considered stronger. Another change in the new version is that if a password is decomposed into multiple chunks, the estimated guess number for each chunk is the larger one between the chunks' original estimated guess number and a min_guess_number, which is 10 if the chunk contains only one character or 50 otherwise. While these changes are heuristic, our experimental results show these changes cause significant improvements under our methods of comparison.

Dell'Amico and Filippone [41] proposed a method to estimate the number of guesses needed to find a password using modern attacks and probabilistic models. Given a probabilistic model, the strength of a passwords is estimated by sampling from the model, i.e., generating random passwords according to the probabilities assigned by the model. This motivates our work to find a way to compare password models, as a better probabilistic model will produce a more accurate estimation.

Recently, Ur et al [22] compared cracking approaches used by researchers with real-world cracking by professionals. They found that semi-automated cracking by professionals outperformed popular fully automated approaches, but could be approximated by combining multiple approaches and assuming the rank of a password was its highest rank among the approaches examined.

2.2 Evaluating User-chosen Passwords and Password Policies

One active research area in recent years is to study the quality of users' password choices under different scenarios, e.g., when facing different password policies [8, 9, 42], when presented with different password strength meters [11, 37], when forced to change passwords due to organizational change of password policies [10], when forced to change passwords due to expiration [12], when creating passwords for different sites [43], when allowed to replace some characters from a randomly generated password [13], and when facing different guidance and feedback [44].

The effect of different password composition policies was studied in [8,9]. They both suggested passwords generated under the policy “Password must have at least 16 characters” provide the most entropy.

In a similar study, Mazurek et al. [42] collected over 25,000 real passwords from CMU and studied passwords from different groups of users. In addition to the difference in passwords from different composition policies, they discovered significant correlations between a number of demographic and behavioral factors and password strength. For example, users associated with the computer science school made passwords more than 1.8 times as strong as those of users associated with the business school.

Egelman et al. [37] examined the impact of password meters on password selection and reported that the presence of meters yielded significantly stronger passwords in a laboratory experiment. However, the meters made no observable difference in a field study when creating passwords for unimportant accounts.

Ur et al. [11] showed that scoring passwords stringently resulted in stronger passwords in general. Additionally, participants who saw stringent password meters spent longer creating their password and were more likely to change their password while entering it, yet they were also more likely to find the password meter annoying. However, the most stringent meter and those without visual bars caused participants to place less importance on satisfying the meter. Participants who saw more lenient meters tried to fill the meter and were averse to choosing passwords a meter deemed “bad” or “poor”.

Shay et al. [10] collected data about behaviors and practices related to the use and creation of passwords from 470 computer users when they were required to change their passwords because of a policy change. They found that, although most of the users were annoyed by the need to create a complex password, they believed that they were more secure.

Zhang et al. [12] investigated the passwords that users chose to replace expired ones. They developed a framework and an efficient algorithm to build an search strategy to search for a user’s new password from an old one. They were able to break future passwords from past ones in 41% of accounts in an offline attack and 17% of accounts in an online attack.

Das et al. [43] investigated how an attacker could leverage a known password from one site to more easily guess that users password at other sites. They estimated that 43-51% of users reuse the same password across multiple sites. For passwords transformed from a basic password between sites, they developed a cross-site password-guessing algorithm, which was able to guess 30% of transformed passwords within 100 attempts.

Huh et al. [13] proposed Surpass, which lets users replace few characters in a random password to make it more memorable. They found that Surpass policies with 3 and 4 character replacements outperformed by 11% to 13% the original randomly-generated password policy in memorability, while showing a small increase in the percentage of cracked passwords.

Shay et al. [44] investigated whether guidance and feedback could enable users to comply with the strict password composition policies imposed by different websites. They showed that password composition policies and password strength meters were not sufficient for getting users to create secure passwords.

In 2014, de Carné de Carnavalet and Mannan [21] examined several password meters in use at selected popular websites, and revealed how the meters worked. They found gross inconsistencies, with the same password resulting in very different strength across different meters.

Komanduri et al. [45] showed that Telepathwords, which makes realtime predictions for the next character that user will type, could help users choosing stronger passwords.

2.3 Password Generation Strategies

The security community has been trying to come up with password generation strategies that can help users generate secure and usable passwords. Candidate strategies have been suggested by sources ranging from the National Institute of Standards and Technology (NIST) [24] to online comics [25], and from security experts' essays [26,27] to online help forums.

Perhaps the most widely recommended and studied strategy is that based on mnemonic sentences: Take a memorable sentence, abbreviate the words, and combine them to form a password. The strategy is generally known as the mnemonic sentence-based strategy (for short, the *mnemonic strategy*). It appears that the general assessment is that this is a good strategy. It is recommended by NIST [24] and by security experts [26, 27]. To our knowledge, three studies on this method have been reported, by Yan et al. [46, 47], Vu et al. [48], and Kuo et al. [49].

Evaluation of the Mnemonic Strategy. Yan et al. [46, 47] conducted a study with college students who were given accounts on a central computing facility. The students were randomly assigned to three groups. The control group (95 members) were asked to create a password with at least seven characters long that contained at least one non-letter. The random password group (96 members) received a sheet of paper with the letters A through Z and the numbers 1 through 9 printed repeatedly on it and were asked to close their eyes and randomly pick eight characters. (They were also advised to keep a written record with them until they had memorized the password.) The mnemonic password group (97 members) were told to create a sentence of 8 words and choose letters from the words to make up a password, mixing upper-case and including at least one non-letter. They found that very few users asked the system administrator to reset their passwords. Responses to an email memorability survey showed that the mnemonic passwords were similar to the control group in terms of difficulty to use, and the random passwords were found to be significantly more difficult. An attack with dictionaries (with permutations with digits) cracked 32% for the control group, 8% for the random password group, and 6% for the mnemonic password group. The authors concluded "*We've debunked another folk belief that random passwords are better than passwords based on mnemonic phrases. In our study, each appeared to be as strong as the other*".

Vu et al. [48] studied two variations of the mnemonic strategy: (A) Choose a sentence containing at least 6 words, and use the first letters from each word as the password; (B) strategy A with an additional requirement that users should embed a special character or digit in the password. Forty psychology students were each asked to create 3 passwords using one of the above strategies. In terms of memorability, they found that participants using strategy B "*took two times longer to recall the passwords, made almost twice as many errors before being able to recall the password, and completely forgot the password twice as often*". Within 12 hours, the L0phtCrack4 (LC4) password cracker cracked all passwords generated with strategy A, whereas only 5% of the passwords from strategy B were cracked.

Kuo et al. [49] conducted a study in which 144 subjects were asked to generate mnemonic passwords, with 146 subjects in the control group. For the control group, they used John the Ripper's 1.2 million-word English dictionary, and were able to crack 11% of the 146 passwords. For the mnemonic group, they collected $129,000$ sentences from the Internet and, with some mangling, created a $400,000$-entry mnemonic password dictionary. Using this dictionary, they cracked 4% of the 144 mnemonic passwords. A bruteforce attack cracked an additional 8% in the control group, and an additional 4% in the mnemonic group. Kuo et al. also searched the Internet (using Google) for the sentences used by the users to generate passwords, and were able to find 65% of them on the Internet. Based on this evidence, the authors concluded that "*Mnemonic phrase-based passwords are not as strong as people may believe, ...*".

We argue that the fact that a password is generated by a sentence that can be found on the Internet does not necessarily mean that it is weak, given that there are likely billions or tens of billions of sentences on Google-indexed pages. Similarly, that a size-$400,000$ dictionary can crack 4% of password seems more like an indicator of strength to us. Using a list of $400,000$ top passwords from Rockyou, one could crack 32% of the passwords in the Yahoo password dataset [50], and 39% of the passwords in the Phpbb dataset [50]. Our interpretation of the data in [49] is that mnemonic sentence-based passwords are significantly stronger than the baseline, as measured by passwords in the Yahoo and phpBB dataset, with two caveats. First, this interpretation is based on cracking results obtained by using their particular dictionary. Second, the conclusion may not be statistically significant because the dataset is small.

Other Related Work. The strategy of choosing 4 random words and concatenating them was made famous by the xkcd comic [25]. Likewise, combining two or three unrelated words and changing some of the letters to numbers or symbols has been recommended by NIST [24]. However, we are unaware of human subject studies on such word-based strategies.

Some have suggested that users should simply use password managers and remember just one password. Password managers, however, create their own security, reliability, and convenience problems [51–55]. Perhaps the biggest concern is that a password manager software takes the security of all critical websites out of the hand of the user and puts it in one piece of software, creating a single point of failure and an attractive target for attackers at the same time. Recently, Xing et al. [56] showed that Unauthorized Cross-App Resource Access (XARA) vulnerabilities on Apple OS X and iOS enable malicious applications to read passwords saved into Apple Keychain and passwords saved in the popular 1Password password manager. These results demonstrate the risk of relying on one password manager for all critical websites.

3. IMPROVING METHODOLOGY TO HELP PASSWORD RESEARCH

A fundamental tool for password security research is that of *probabilistic password models* (*password models* for short).

A probabilistic password model assigns a probability value to each string. Such models are useful for research into understanding what makes users choose more (or less) secure passwords, and for constructing password strength meters and password cracking utilities. Guess number graphs generated from password models are a widely used method in password research. In this chapter, we show that probability-threshold graphs have important advantages over guess-number graphs. They are much faster to compute, and at the same time provide information beyond what is feasible in guess-number graphs. We also observe that research in password modeling can benefit from the extensive literature in statistical language modeling. We conduct a systematic evaluation of a large number of probabilistic password models, including Markov models using different normalization and smoothing methods, and find that, among other things, Markov models, when done correctly, perform significantly better than the Probabilistic Context-Free Grammar model proposed in Weir et al. [18], which has been used as the state-of-the-art password model in recent research.

3.1 Introduction

A password model assigns a probability value to each string. The goal of such a model is to approximate as accurately as possible an unknown password distribution $\mathcal{D}$. We divide password models into two classes, *whole-string models* and *template-based models*. A template-based model divides a password into several segments, often by grouping consecutive characters of the same category (e.g., lower-case letters, digits, etc.) into one segment, and then generates the probability for each segment independently, e.g., [17, 18]. A whole-string model, on the other hand, does not divide a password into segments, e.g., the Markov chain model in [19].

We classify research involving password models into two types. *Type-1* research aims at understanding what makes users choose more (or less) secure passwords. To do this, one obtains password datasets chosen by users under difference circumstances, and then uses a password model to compare the relative strengths of these sets of passwords. *Type-2* research aims at finding the best password models. Such a model can then be used to evaluate the level of security of a chosen password, as in password strength meters. Such a model can also be used to develop more effective password cracking utilities, as a password model naturally determines the order in which one should make guesses.

Type-1 research has been quite active in recent years. Researchers have studied the quality of users' password choices under different scenarios [8–12]. In this area, earlier work uses either standard password cracking tools such as John the Ripper (JTR) [14], or ad hoc approaches for estimating the information entropy of a set of passwords. One such approach is NIST's recommended scheme [15] for estimating the entropy of one password, which is mainly based on their length.

Weir et al. [18] developed a template-based password model that uses Probabilistic Context-free Grammar. We use PCFG_W to denote the model is this chapter. In [16], they argued that entropy estimation methods such as that recommended by NIST were inaccurate metrics for password strength, and proposed to use the guess numbers of passwords. The guess number of a password according to a password model is defined to be the rank of the password in the order of decreasing probability. To compare two sets of passwords, one plots the number of guesses vs. the percentage of passwords cracked by the corresponding number of guesses in the testing dataset. Such *guess-number graphs* are currently the standard tool in password research. Plotting such graphs, however, requires the computation of guess numbers of passwords, which is computationally expensive. For many password models, this requires generating all passwords with probability above a certain threshold and sorting them. Some template-based models, such as the PCFG_W model, have the property that all strings fitting a template are assigned the same probability. One is thus able to compute guess numbers by maintaining information about templates instead of individual passwords. This is done in the guess calculator framework in [8, 42], which is based on the PCFG$_W$ model. This framework uses parallel computation to speed up the process, and is able to go up to $\approx 4E14$ [42]. Even with this approach, however, one is often limited by the search space so that only a portion (typically between 30% and 60%) of the passwords are covered [8, 42]. Such studies thus miss information regarding the stronger passwords.

We observe that for type-1 research, in which one compares the relative strength of two sets of passwords, it is actually unnecessary to compute the guess numbers of all passwords. It suffices to compute the probabilities of all passwords in the datasets, which can be done much faster, since the number of passwords one wants to evaluate (often much less than 1E6) is in general extremely small compared to the maximum guess number one is interested in. We propose to plot the probability threshold vs. the percentage of passwords cracked curves, which we call *probability-threshold* graphs. In addition to being much faster to compute, another advantage is that such a curve shows the quality of passwords in the set all the way to the point when all passwords are cracked (assuming that the password model assigns a non-zero probability to each allowed password).

A natural approach for password models is to use whole-string Markov chains. Markov chains are used in the template-based model in [17], for assigning probabilities to segments that consists of letters. Castelluccia et al. [19] proposed to use whole-string Markov models for evaluating password strengths, without comparing these models with other models. At the time of this chapter's writing, PCFG$_W$ was considered to be the model of choice in type-1 password research.

We find this current state of art unsatisfying. First, only very simple Markov models have been considered. Markov models are known as n-gram models in the statistical language literature, and there exist a large number of techniques developed to improve the performance of such models. In particular, many smoothing techniques were developed to help solve the sparsity and overfitting problem in higher-order Markov models. Such techniques include Laplace smoothing, Good-Turing smoothing [57], and backoff [58]. Password modeling research has not taken advantage of such knowledge and techniques.

Second, passwords differ from statistical language modeling in that passwords have a very definite length distributions: most passwords are between lengths 6 and 12, and there are very few short and long passwords. The n-gram models used in statistical language modeling generally have the property that the probabilities assigned to all strings of a fixed length add up to 1, implicitly assuming that the length distribution is uniform. This property is fine for natural language applications, because often times one only needs to compare probabilities of sentences of the same (or very similar) length(s), e.g., when determining which sentence is the most likely one when given a sequence of sounds in speech recognition. For password models, however, assuming uniform length distribution is suboptimal. Finally, different password modeling approaches have not been systematically evaluated or compared against each other. In particular, a rigorous comparison of the PCFG$_W$ model with whole-string Markov models has not been performed.

In this chapter, we conduct an extensive empirical study of different password models using 6 real-world plaintext password datasets totaling about 60 million passwords, including 3 from American websites and 3 from Chinese websites. We consider three different normalization approaches: *direct*, which assumes that strings of all lengths are equally likely, *distribution-based*, which uses the length distribution in the training dataset, and *end-symbol based*, which appends an “end” symbol to each password for training and testing. We consider Markov chain models of different orders, and with different smoothing techniques. We compare whole-string models with different instantiation of template-based models.

Some of the important findings are as follows. First and foremost, whole-string Markov models, when done correctly, significantly outperform the PCFG$_W$ model [18] when one goes beyond the first million or so guesses. PCFG$_W$ uses as input both a training dataset, from which it learns the distribution of different templates, and a dictionary from which it chooses words to instantiate segments consisting of letters. In this chapter, we consider 3 instantiations of PCFG$_W$: the first uses the dictionary used in [18]; the second uses the OpenWall dictionary; and the third generates the dictionary from the training set. We find that the third approach works significantly better than the first and the second; in addition, all three instantiations significantly underperform the best whole-string Markov models. Furthermore, higher orders of Markov chains show different degrees of overfitting effect. That is, they perform well for cracking the high-probability passwords, but less so later on. The backoff approach, which essentially uses a variable-order Markov chain, when combined with end-symbol based normalization, suffers little from the overfitting effect and performs the best.

3.2 Password Models and Metrics

In this section, we introduce probability-threshold graphs.

We use Σ to denote the alphabet of characters that can be used in passwords. We further assume that all passwords are required to be of length between L and U for some values of L and U; thus the set of allowed passwords is

$$\Gamma = \bigcup_{\ell=L}^{U} \Sigma^{\ell}.$$

In the experiments in this chapter, we set Σ to include the 95 printable ASCII characters, and $L = 4$ and $U = 40$. Sometimes the alphabet Σ is divided into several subsets, which we call the character categories. For example, one common approach is to divide the 95 characters into four categories: lower-case letters, upper-case letters, digits, and special symbols.

Definition 3.2.1 *A* password probability model *(password model for short) is given by a function* p *that assigns a probability to each allowed password. That is, a password model* $p : \Gamma \to [0, 1]$ *satisfies*

$$\forall_{s\in\Gamma}\, p(s) \geq 0 \bigwedge \sum_{s\in\Gamma} p(s) = 1.$$

We say that a password model p *is* complete *if and only if it assigns a non-zero probability to any string in* Γ.

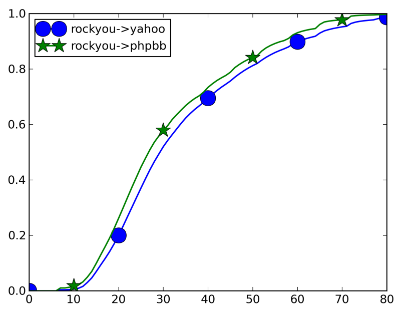
For a password model p to be useful in practice, it should be *efficiently computable*; that is, for any $s \in \Gamma$, computing $p(s)$ takes time proportional to $O(|\Sigma| \cdot \mathit{length}(s))$. For p to be useful in cracking a password, it should be *efficiently enumerable*; that is, for any integer N, it runs in time proportional to $O(N \cdot |\Sigma| \cdot U)$ to output the N passwords that have the highest probabilities according to p. If one desires to make a large number of guesses, then the password generation also needs to be *space efficient*, i.e., the space used for generating N passwords should grow at most sub-linearly in N. Ideally, the space usage (not counting the generated passwords) should be independent from N.

We consider the following three methods/metrics when running a given password model with a testing password dataset. Note that for type-1 research, we fix the model and compare different datasets. For type-2 research, we fix a dataset and compare different models.

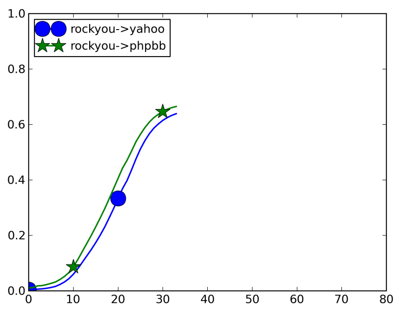
Guess-number graphs. Such a graph plots the number of guesses in log scale vs. the percentage of passwords cracked in the dataset. A point (x, y) on a curve means that y percent of passwords are included in the first 2^x guesses (i.e., the 2^x passwords that have the highest probabilities). This approach is logically appealing; however, it is also highly computationally expensive, since it requires generating a very large number of password guesses in decreasing probability. This requirement makes it difficult when we want to compare many different approaches. Furthermore, because of the resource limitation, we are unable to see the effect beyond the guesses we have generated. These limitations motivate us to use the following metrics.

Probability-threshold graphs. Such a graph plots the probability threshold in log scale vs. the percentage of passwords above the threshold. A point (x, y) on a curve means that y percent of passwords in the dataset have probability at least 2^{-x}. To generate such a figure, one needs only to compute the probabilities the model assigns to each password in the testing dataset. For each probability threshold, one counts how many passwords are assigned probabilities above the threshold. For a complete password model, such a curve can show the effect of a password model on the dataset all the way to the point when all passwords in the dataset are included.

Figure 3.1 shows a probability-threshold graph and the corresponding guess number graph, for comparing the relative strength of two datasets Phpbb and Yahoo using a Markov chain model of order 5 trained on the Rockyou dataset.



(a) Probability-threshold graph



(b) Guess-number graph

Figure 3.1.: An example of probability-threshold graph and guess-number graph

An interesting question is how a probability-threshold graph relates to the corresponding guess-number graph. If one draws conclusions regarding two curves based on the former, do the conclusions hold in the latter?

When comparing the strength of two password datasets using a fixed model, the answer is “yes”. The relationship of two guess-number curves (e.g., which curve is to the left of the other, whether and when they cross, etc.) would be exactly the same as that demonstrated by the beginning portions of the corresponding probability-threshold curves; because for a given threshold, exactly the same set of passwords will be attempted. This effect can be observed from Figure 3.1. The only information missing is that we do not know the exact guess number corresponding to each probability threshold; however, this information is not needed when our goal is to compare two datasets. In addition to being much faster to compute, the probability-threshold graph is able to show the quality of passwords in the set all the way to the point when all passwords are cracked (assuming that the password model assigns a non-zero probability to each allowed password).

When comparing models with a fixed dataset, in general, the answer is no. In the extreme case, consider a model that assigns almost the same yet slightly different probabilities to every possible password in Γ, while using the same ranking as in the testing dataset. As a result, the probabilities assigned to all passwords would all be very small. Such an unrealistic model would perform very well in the guess-number graph; however, the probability-threshold graph would show extremely poor performance, since no password is cracked until a very low probability threshold is reached.

When comparing two password models, whether the conclusions drawn from a probability threshold graph can be carried over to the guess number graph depends on whether the two models have similar rank distributions. Given a password model p, the rank distribution gives for each $i \in \{1, 2, 3, \cdots\}$, the probability of the password that has the i'th highest probability. Thus, if two password models are known to generate very similar rank distributions, then any conclusion drawn from the probability-threshold graph will also hold on the guess number graph, because the same probability threshold will correspond to the same guess number. When two models generate somewhat different rank distributions, then one needs to be careful when interpreting results obtained from probability-threshold graphs. In particular, one may want to also generate the guess number graphs to check whether the results are similar at least for the beginning portion of the probability-threshold graphs. There is the main limitation of using probability-threshold graphs in type-2 research.

Average-Negative-Log-Likelihood (ANLL) and ANLL$_\theta$. Information encoded in a probability-threshold curve can be summarized by a single number that measures the area to the left of the curve, which turns out to be exactly the same as the Average Negative Log Likelihood (ANLL). ANLL has its root in statistical language modeling. A statistical language model assigns a probability to a sentence (i.e., a sequence of words) by means of a probability distribution. Language modeling is used in many natural language processing applications such as speech recognition, machine translation, part-of-speech tagging, parsing and information retrieval. Password modeling can be similarly viewed as trying to approximate as accurately as possible $\mathcal{D}$, an unknown distribution of passwords. We do not know $\mathcal{D}$, and instead have a testing dataset D, which can be viewed as sampled from $\mathcal{D}$. We use p_D to denote the probability distribution given by the dataset D. Representing the testing dataset D as a multi-set $\{s_1, s_2, \ldots, s_n\}$, where a password may appear more than once, the ANLL metric is computed as follows:

$$\mathsf{ANLL}(D|p) = \frac{1}{|D|} \sum_{s \in D} -\log_2 p(s)$$

To see that ANLL equals the area to the left of the probability-threshold curve, observe that when dividing the area into very thin horizontal rows, the length of a row with height between y and $y + dy$ is given by the number of passwords with probability p such that $y \leq -\log_2 p < y + dy$. By summing these areas up, one obtains the ANLL.

ANLL gives the same information as *Empirical Perplexity*, which is the most widely used metric for evaluating statistical language models, and is defined as $2^{\mathsf{ANLL}(D|p)}$.

While using a single number to summarize the information in a probability-threshold curve is attractive, obviously one is going to lose some information. As a result, ANLL has some limitations. First, ANLL is applicable only for complete password models, i.e., those that assign a non-zero probability to each password. In the area interpretation, if a password in the dataset is assigned probability 0, it is never guessed, and the curve never reaches $Y = 1$, resulting in an infinite area. Second, ANLL includes information about cracking 100 percent of the passwords, which may not be what one wants. Since it may be infeasible to generate enough guesses to crack the most difficult passwords, one may care about only the θ portion of passwords that are easy to crack. In that case, one could use ANLL$_\theta$, which we define to be the area to the left of the curve below θ. In this chapter, we use ANLL$_{0.8}$ when comparing different models using one single number.

3.3 Design Space of Password Models

In this section, we explore the design space of password models.

We mostly consider approaches that construct password models without relying on an external dictionary. That is, we consider approaches that can take a training password dataset and learn a password model from it. The performance of approaches that rely on external dictionaries depends very much on the quality of the dictionaries, and in particular, how well the dictionaries match the testing dataset. Such approaches are not broadly applicable, and it is difficult to evaluate their effectiveness.

N-gram models, i.e., Markov chains, are the dominant approach for statistical languages. It is natural to apply them to passwords. A Markov chain of order d, where d is a positive integer, is a process that satisfies

$$P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) = P(x_i|x_{i-1}, \ldots, x_{i-d})$$

where d is finite and $x_1, x_2, x_3, \ldots$ is a sequence of random variables. A Markov chain with order d corresponds to an n-gram model with $n = d + 1$.

We divide password models into two classes, *whole-string models* and *template-based models*. A template-based model divides a password into several segments, often by grouping consecutive characters of the same category into one segment, and then generates the probability for each segment independently. A whole-string model, on the other hand, does not divide a password into segments.

3.3.1 Whole-String Markov Models

Whole-string Markov models are used in John the Ripper (JTR) [14] and Castelluccia et al.'s work on password strength estimation [19]. JTR in Markov mode uses a 1-order Markov chain, in which the probability assigned to a password "$c_1 c_2 \cdots c_\ell$" is

$$P(\text{"}c_1 c_2 \cdots c_\ell\text{"}) = P(c_1|c_0)P(c_2|c_1)P(c_3|c_2) \cdots P(c_\ell|c_{\ell-1}),$$

where $c_0 \notin \Sigma$ denotes a start symbol that is prepended to the beginning of all passwords, and

$$P(c_i|c_{i-1}) = \frac{count(c_{i-1}c_i)}{count(c_{i-1}\cdot)} \tag{3.1}$$

where $count(c_{i-1}\cdot)$ denotes the number of occurrences of c_{i-1} where it is followed by another character (i.e., where it is not at the end of password), and $count(c_{i-1}c_i)$ gives the number of occurrences of the substring $c_{i-1}c_i$. By definition, we have $count(c_{i-1}\cdot) =$

$\sum_{c_i \in \Sigma} count(c_{i-1}c_i)$. When we use Markov chains of order $d > 1$, we can prepend d copies of the start symbol c_0 to each password.

There are many variants of whole-string Markov models. Below we examine the design space.

Need for Normalization. We note that models used in [14, 19] are not probability models because the values assigned to all strings do not add up to 1. In fact, they add up to $U-L+1$, where U is the largest allowed password length, and L is the smallest allowed password length, because the values assigned to all strings of a fixed length add up to 1. To see this, first observe that

$$\sum_{c_1 \in \Sigma} P(\text{``}c_1\text{''}) = \frac{\sum_{c_1 \in \Sigma} count(c_0 c_1)}{count(c_0 \cdot)} = 1,$$

and thus the values assigned to all length-1 strings add up to 1. Assume that the values assigned to all length-ℓ strings sum up to 1. We then show that the same is the case for length $\ell + 1$ as follows.

$$\begin{aligned}
& \textstyle\sum_{c_1 c_2 \cdots c_{\ell+1} \in \Sigma^{\ell+1}} P(\text{``}c_1 c_2 \cdots c_{\ell+1}\text{''}) \\
= & \textstyle\sum_{c_1 c_2 \cdots c_\ell \in \Sigma^\ell} P(\text{``}c_1 c_2 \cdots c_\ell\text{''}) \times \sum_{c_{\ell+1} \in \Sigma} P(c_{\ell+1} | c_\ell) \\
= & \textstyle\sum_{c_1 c_2 \cdots c_\ell \in \Sigma^\ell} P(\text{``}c_1 c_2 \cdots c_\ell\text{''}) = 1
\end{aligned}$$

The same analysis holds for Markov chains of order $d > 1$. To turn such models into probability models, several normalization approaches can be applied:

Direct normalization. The approach of using the values directly in [14, 19] is equivalent to dividing each value by $(U - L + 1)$. This method, however, more or less assumes that the total probabilities of passwords for each length are the same. This assumption is clearly not the case in practice. From Table 3.6 and Table 3.7, which give the password length distributions for the datasets we use in this chapter, we can see that passwords of lengths between 6 and 10 constitute around 87% of all passwords, and passwords of lengths 11 and 12 add an additional 7%, with the remaining 30 lengths (lengths 4,5,13-40) together contributing slightly less than 6%.

Distribution-based normalization. A natural alternative to direct normalization is to normalize based on the distribution of passwords of different lengths. More specifically, one normalizes by multiplying the value assigned to each password of length m with the following factor:

$$\frac{\text{\# of passwords of length } m \text{ in training}}{\text{total \# of passwords in training}}$$

This approach, however, may result in overfitting, since the training and testing datasets may have different length distributions. From Table 3.7, we can see that the CSDN dataset includes 9.78% of passwords of length 11, whereas the PhpBB dataset includes only 2.1%, resulting in a ratio of 4.66 to 1, and this ratio keeps increasing. At length 14, CSDN has 2.41%, while PhpBB has only 0.21%.

End-symbol normalization. We propose another normalization approach, which appends an "end" symbol c_e to each password, both in training and in testing. The probability assigned by an order-1 Markov chain model to a password "$c_1 c_2 \cdots c_\ell$" is

$$P(c_1|c_0)P(c_2|c_1)P(c_3|c_2)\cdots P(c_\ell|c_{\ell-1})P(c_e|c_\ell),$$

where the probability of $P(c_e|c_\ell)$ is learned from the training dataset. For a string "$c_1 c_2 \cdots c_U$" of the maximal allowed length, we define $P(c_e|c_U) = 1$.

In this approach, we also consider which substrings are more likely to occur at the end of passwords. In Markov chain models using direct normalization, the prefix of a password is always more likely than the actual password, which may be undesirable. For example, "passwor" would be assigned a higher probability than "password". The end symbol corrects this situation, since the probability that an end symbol following "passwor" is likely to be significantly lower than the product of the probability that "d" follows "passwor" and that of the end symbol following "password".

Such a method can ensure that the probabilities assigned to all strings sum up to 1 when the order of the Markov chain is at least as large as L, the smallest allowed password length.

To see this, consider the case when $L = 1$. Envision a tree where each string corresponds to a node in the tree, with c_0 being the root node. Each string "$c_0c_1 \cdots c_\ell$" has "$c_0c_1 \cdots c_{\ell-1}$" as its immediate parent. All the leaf nodes are strings with c_e as the last symbol, i.e., they are actual passwords. Each edge is assigned the transition probability; thus, the values assigned to all edges leaving a parent node add up to 1. Each node in the tree is assigned a value. The root is assigned 1, and every other node is assigned the value of the parent multiplied by the transition probability of the edge from its parent to the node. Thus, the value assigned to each leaf node equals its probability under the model. We note that the value assigned to each node equals the sum of values assigned to its children. It follows that the total probabilities assigned to all leaves add up to be the same as those assigned to the root, which is 1.

When the order of Markov chain, d, is less than L, the minimal required password length, we may have the situation that the total probabilities add up to less than 1, because non-zero probabilities would be assigned to illegal passwords of lengths less than L. The effect is that this method is slightly penalized when evaluated using probability-threshold graphs or ANLLs. We note that when a model wastes half of the total probability of 1 by assigning them to passwords that are too short, the ANLL$_\theta$ is increased by θ.

Choice of Markov Chain Order, Smoothing, and Sparsity. Another challenging issue is the choice of the order for Markov chains. Intuitively, a higher-order Markov chain enables the use of deeper history information than lower-order ones, and thus may be more accurate. However, at the same time, a higher-order Markov chain runs the risk of overfitting and sparsity, which causes reduced performance. Sparsity means that one is computing transition probabilities from very small count numbers, which may be noisy.

In Markov chain models, especially higher order ones, many calculated conditional probabilities would be 0, causing many strings assigned probability 0. This can be avoided by applying the technique of smoothing, which assigns pseudocounts to unseen strings. Pseudocounts are generally motivated on Bayesian grounds. Intuitively, one should not think that things that one has not yet seen have probability 0. Various smoothing methods have been developed for language modeling. Additive smoothing, also known as Laplace smoothing, adds δ to the count of each substring. When δ is chosen to be 1, this is known as add-one smoothing, which is problematic because the added pseudo counts often overwhelm the true counts. In this chapter, we choose δ to be 0.01; the intuition is that when encountering a history "$c_1c_2c_3c_4$", all the unseen next characters, after smoothing come up to slightly less 1, since there are 95 characters.

A more sophisticated smoothing approach is Good-Turing smoothing [57]. This was developed by Alan Turing and his assistant I.J. Good as part of their efforts at Bletchley Park to crack German Enigma machine ciphers during World War II. An example illustrating the effect of Good-Turing smoothing is in [59], where Bonneau calculated the result of applying Good-Turing smoothing to the Rockyou dataset. After smoothing, a password that appears only once has a reduced count of 0.22, a password that appears twice has a count of 0.88, whereas a password that appears $k \geq 3$ times has a reduced count of approximately $k - 1$. The "saved" counts from these reduction are assigned to all unseen passwords. Intuitively, estimating the probability of a password that appears only a few times using its actual count will overestimate its true probability, whereas the probability of a password that appears many times can be well approximated by its probability in the dataset. The key observation underlying Good Turing smoothing is that the best estimation for the total probability of items that appear exactly i times is the total probability of items that appear exactly $i + 1$ times. In particular, the total probability for all items unseen in a dataset is best estimated by the total probability of items that appear only once.

Grouping. Castelluccia et al. [19] proposed another approach to deal with sparsity. They observed that upper-case letters and special symbols occur rarely, and propose to group them into two new symbols. Here we use Υ for the 26 upper-case letters and Ω for the 33 special symbols. This approach reduces the size of the alphabet from 95 to 38 (26 lower case letters, 10 digits, and Υ and Ω), thereby reducing sparsity. It treats all uppercase letters as exactly the same, and all special symbols as exactly the same. That is, the probability that any upper-case letter follows a prefix is the probability that Υ follows the prefix divided by 26. For example, $P[$"aF?b"$]$ for an order 2 model is given by

$$P[\text{a}|s_0s_0]\,\frac{P[\Upsilon|s_0\text{a}]}{26}\,\frac{P[\Omega|\text{a}\Upsilon]}{33}\,P[b|\Upsilon\Omega].$$

We call this the "grouping" method.

We experimented with an adaptation of this method. When encountering an upper-case letter, we assign probabilities in proportion to the probability of the corresponding lower-case letter, based on the intuition that following "swor", "D" is much more likely than Q, just as d is much more like than q. When encountering a special symbol, we use the order-1 history to assign probability proportionally, based on the intuition that as special symbols are rare, a shorter history is likely better. In the adaptation, the probability of the above string will be computed as:

$$P[\text{a}|s_0s_0]\,\frac{P[\Upsilon|s_0\text{a}]P[f|s_0\text{a}]}{\sum_\alpha P[\alpha|s_0\text{a}]}\,\frac{P[\Omega|a\Upsilon]P[?|F]}{\sum_\omega P[\omega|F]}\,P[b|\Upsilon\Omega],$$

where α ranges over all lower-case letters, and ω ranges over all special symbols.

Backoff. Another approach that addresses the issue of order, smoothing, and sparsity together is to use variable order Markov chains. The intuition is that if a history appears frequently, then we would want to use that to estimate the probability of the next character. For example, if the prefix is "passwor", using the whole prefix to estimate the next character would be more accurate than using only "wor". On the other hand, if we observe a prefix that rarely appears, e.g., "!W}ab", using only "ab" as the prefix is likely to be more accurate than using longer history, which likely would assign equal probability to all characters. One way to do this is Katz's backoff model [58]. In this model, one chooses a threshold and stores all substrings whose counts are above the threshold. Let $\pi_{i,j}$ denote the substring of $s_0s_1s_2\cdots s_\ell$ from s_i to s_j. To compute the transition probability from $\pi_{0,\ell-1}$ to $\pi_{0,\ell}$, we look for the smallest i value such that $\pi_{i,\ell-1}$ has a count above the threshold. There are two cases. In case one, $\pi_{i,\ell}$'s count is also above the threshold; thus, the transition probability is simply $\frac{count(\pi_{i,\ell})}{count(\pi_{i,\ell-1})}$. In case two, where $\pi_{i,\ell}$ does not have a count, we have to rely on a shorter history to assign this probability. We first compute b, the probability left after assigning transition probabilities to all characters via case one. We then compute the probabilities for all other characters using the shorter prefix $\pi_{i+1,\ell-1}$, which are computed in a recursive fashion. Finally, we normalize these probabilities so that they add up to b.

Backoff models are significantly slower than other Markov models. The backoff models used in experimentation are approximately 11 times slower than the plain Markov models, both for password generation and for probability estimation.

3.3.2 Template-based Models

In the template-based approach, one divides passwords into different templates based on the categories of the characters. For example, one template is 6 lower-case letters followed by 3 digits, which we denote as $\alpha^6\beta^3$. Such a template has two segments, one consisting of the first 6 letters and the other consisting of last 3 digits. The probability of the password using this template is computed as:

$$P(\text{“passwd123”}) = P(\alpha^6\beta^3)P(\text{“passwd”}|\alpha^6)P(\text{“123”}|\beta^3),$$

where $P(\alpha^6\beta^3)$ gives the probability of the template $\alpha^6\beta^3$, $P(\text{“passwd”}|\alpha^6)$ gives the probability that “passwd” is used to instantiate 6 lowercase letters, and $P(\text{“123”}|\beta^3)$ gives the probability that “123” is used to instantiate 3 digits. To define a model, one needs to specify how to assign probabilities to templates, and to the different possible instantiations of a given segment. Template-based models are used in [17] and the PCFG$_W$ model [18].

Assigning probabilities to templates. In [17], the probabilities for templates are manually chosen because this work was done before the availability of large password datasets. In [18], the probability is learned from the training dataset by counting. That is, the probability assigned to each template is simply the number of passwords using the template divided by the total number of passwords. Using this approach means that passwords that use a template that does not occur in the training dataset will have probability 0.

We consider the alternative of applying Markov models to assign probabilities to templates. That is, we convert passwords to strings over an alphabet of $\{\alpha, \beta, \upsilon, \omega\}$, representing lower-case letters, digits, upper-case letters, and symbols respectively. We then learn a Markov model over this alphabet, and assign probabilities to a template using Markov models.

Assigning probabilities to segments. In [17], the probabilities of letter segments are assigned using a Markov model learned over natural language, and the probabilities for digit segments and symbol segments are assigned by assuming that all possibilities are equally likely. In the PCFG$_W$ model [18], the probabilities for digit segments and symbol segments are assigned using counting from the training dataset, and letter segments are handled using a dictionary. That is, for an α^6 segment, all length-6 words in the dictionary are assigned an equal probability, and any other letter sequence is assigned probability 0. For a β^3 template, the probability of "123" is the number of times "123" occurs as a 3-digit segment in the training dataset divided by the number of 3-digit segments in the training dataset.

We consider template models in which we assign segment instantiation probabilities via both counting and Markov models. Table 3.1 summarizes the design space of password models we consider in this chapter.

PCFG$_W$ model does not fit in the models in Table 3.1, as it requires as input a dictionary in addition to a training dataset. In this chapter, we considered 3 instantiations of PCFG$_W$: the first uses the dictionary used in [18]; the second uses the OpenWall dictionary; and the third generates the dictionary from the training set. We note that the last instantiation is essentially the template-based model using counting both for assigning template probabilities and for instantiating segments.

Table 3.1.: Design space for probabilistic password models

Whole-string Markov models	normalization methods: (1) direct, (2) distribution-based, (3) end symbol order of Markov chain: $1, 2, 3, 4, \cdots$ or variable (backoff) dealing with sparsity: (1) plain, (2) grouping (3) adapted grouping smoothing methods: (1) no smoothing, (2) add-δ smoothing, (3) Good-Turing smoothing
Template-based models	template probability assignment: (1) Counting, (2) Markov model segment probability assignment: (1) Counting, (2) Markov model

3.3.3 Password Generation

To use a password model for cracking, one needs to be able to generate passwords in decreasing probability.

In whole-string Markov-based methods, the password search space can be modeled as a search tree. As described earlier, the root node represents the empty string (beginning of the password), and every other node represents a string. Each node is labeled with the probability of the string it represents.

One algorithm for password guess generation involves storing nodes in a priority queue. The queue, arranged in order of node probabilities, initially contains the root node of the search tree. We then iterate through the queue, popping the node with the greatest likelihood at each iteration. If this is an end node (any non-root node for uniform or length-based normalization, or nodes with end symbol for end-symbol normalization), we output the string as a password guess. Otherwise, we calculate the transition probability of each character in the character set and add to the priority queue a new child node with that character and the associated probability. This algorithm is guaranteed to return password guesses in decreasing order of probability (as estimated by the model), due to the iteration order and property that each node's probability is smaller than that of its parent. The algorithm terminates once the desired number of guesses has been generated.

The priority-queue method, however, is not memory efficient, and does not scale for generating a large number (e.g., over 50 million) of guesses. Since each node popped from the queue can result in multiple nodes added to the queue, the queue size is typically several times of the number of guesses generated. To reduce the memory footprint, we propose a threshold search algorithm.

The threshold search algorithm, similar to the iterative deepening state space search method, conducts multiple depth-first traversals of the search tree. In the i'th iteration, it generates passwords with probabilities in a range $(\rho_i, \tau_i]$, by performing a depth-first traversal of the tree, pruning all nodes with probability less than ρ_i, and outputting all passwords with probability in the target range. To generate n password guesses, we start with a conservative range of $(\rho_1 = \frac{1}{n}, \tau_1 = 1]$. After the i'th iteration, if we have generated $m < n$ passwords so far, we start another iteration with $(\rho_{i+1} = \frac{\rho_i}{\max(2,1.5n/m)}, \tau_{i+1} = \rho_i]$. That is, when $m < 0.75n$, we halve the probability threshold. We have observed empirically that halving ρ result in close to twice as many passwords being generated. We may overshoot and generally more than n passwords, but are very unlikely to generate over $2n$ guesses. The average runtime complexity of this algorithm as $O(n)$, or linear on n. The memory complexity (not including the model data or generated passwords) is essentially constant, as the only data structure needed is a stack of capacity $U + 1$. We use this framework to efficiently generate Markov model-based guesses in the experiments. Slight adjustments need to be made for distribution-based normalization. This method, however, does not apply to template-based models. Through probability-threshold graphs, we have found that unless Markov models are used to instantiate templates, template-based models perform rather poorly. However, when Markov models are used, efficient generation for very large numbers of passwords appears quite difficult, as one cannot conduct depth-first search and needs to maintain a large amount of information for each segment. In this chapter, we do not do password generation for template-based models other than the PCFG$_W$ model.

3.4 Experimental Methodologies

In this section, we describe our experimental evaluation methodologies, including the dataset we use and the choice of training/testing scenarios.

Datasets. We use the following six password datasets downloaded from public Web sites. We use only the password information in these datasets and ignored all other information (such as user names and/or email addresses included in some datasets). The **RockYou** dataset [50] contains over 32 million passwords leaked from the social application site Rockyou in December 2009. The **PhpBB** dataset [50] includes about 250K passwords cracked by Brandon Enright from MD5 hashes leaked from Phpbb.com in January 2009. The **Yahoo** dataset includes around 450K passwords published by the hacker group named D33Ds in July 2012. The **CSDN** dataset includes about 6 million user accounts for the Chinese Software Developer Network (CSDN), a popular IT portal and a community for software developers in China [60]. The **Duduniu** dataset includes about 16 million accounts for a paid online gaming site in China. The **178** datasets includes about 10 million passwords for another gaming website in China. All 3 Chinese dataset were leaked in December 2011.

The first 3 datasets are from American websites, and the last 3 are from Chinese websites. The 6 datasets together have approximately 60 million passwords.

Data cleansing. We performed the following data cleansing operations. First, we removed any password that includes characters beyond the 95 printable ASCII symbols. This step removed 0.034% of all passwords. In the second step, we removed passwords whose length were less than 4 or greater than 40. As almost any system that uses passwords will have a minimal length requirement and a maximum length restriction, we believe that such processing is reasonable. In total we removed 0.037% of passwords that are too short, and 0.002% of passwords that are too long. Length-4 passwords account for 0.225% of the datasets, and we chose not to remove them. Table 3.2 gives detailed information on cleansing.

Table 3.2.: Results of removing passwords that are too long, too short, and non-ASCII passwords.

	Size after cleansing		Removed						
Dataset	**Unique**	**Total**	**non-ASCII Unique**	**non-ASCII Total**	**too-short Unique**	**too-short Total**	**too-long Unique**	**too-long Total**	**Percentage of All Removed**
RockYou	14325681	32575584	14541	18038	2868	7914	1290	1346	0.08%
PhpBB	183400	253512	45	45	944	1864	0	0	0.75%
Yahoo	342197	442288	0	0	283	497	0	0	0.11%
CSDN	4037139	6427522	314	355	465	754	0	0	0.02%
Duduniu	6930996	10978339	1485	1979	3685	10881	28	28	0.12%
178	3462280	9072960	0	0	3	5	1	1	0.00%

Dataset statistics. Table 3.3 shows the percentages of passwords that appeared $1, 2, 3, 4, 5+$ times in all datasets. The detailed statistics for American datasets and Chinese datasets are listed in Table 3.4 and Table 3.5, respectively. Recall that the Good-Turing method estimates that the total probabilities of unseen passwords to be that of unique passwords. We see that the two smallest datasets, PhpBB and Yahoo, have significantly higher percentages of unique passwords, 64.16% and 69.83% respectively, compared to 36.43% for Rockyou. When combining all 6 datasets, approximately 40% are unique.

Table 3.3.: Password count and frequency information of all datasets

	All
all	59750205
Percentage due to Unique	40.92%
Percentage due to Twice	9.32%
Percentage due to 3 Times	4.02%
Percentage due to 4 Times	2.36%
Percentage due to 5+ Times	43.38%

Table 3.4.: Password count and frequency information of American datasets

	All American	**RockYou**	**PhpBB**	**Yahoo**
all	33271384	32575584	253512	442288
Percentage due to Unique	37.09%	36.43%	64.16%	69.83
Percentage due to Twice	8.15%	8.12%	9.79%	9.38%
Percentage due to 3 Times	3.58%	3.59%	3.86%	3.60%
Percentage due to 4 Times	2.25%	2.25%	2.31%	1.94%
Percentage due to 5+ Times	48.93%	49.61%	19.88%	15.25%

Table 3.6 and Table Table 3.7 show the length distributions of the password datasets from English users and from Chinese users, respectively. The tables suggest that the most common password lengths are 6 to 10, which account for 87% of the whole dataset. One interesting aspect is that the CSDN dataset has much fewer length 6 and 7 passwords than other datasets. One explanation is that the website started enforcing a minimal length-

Table 3.5.: Password count and frequency information of Chinese datasets

	All Chinese	CSDN	Duduniu	178
all	26478821	6427522	10978339	9072960
Percentage due to Unique	45.74%	55.72%	51.83%	31.30%
Percentage due to Twice	10.73%	10.79%	14.20%	7.80%
Percentage due to 3 Times	4.46%	4.55%	5.96%	3.65%
Percentage due to 4 Times	2.47%	2.51%	3.18%	2.19%
Percentage due to 5+ Times	36.41%	29.84%	24.83%	55.07%

Table 3.6.: Password length information for English datasets

	All American	RockYou	PhpBB	Yahoo
4	0.2444%	0.2164%	3.1833%	0.6215%
5	4.0466%	4.0722%	5.7110%	1.2028%
6	25.9586%	26.0553%	27.4212%	17.9994%
7	19.2259%	19.2966%	17.8169%	14.8313%
8	20.1374%	19.9886%	27.3967%	26.9336%
9	12.1343%	12.1198%	9.1562%	14.9125%
10	9.0805%	9.0650%	5.3276%	12.3795%
11	3.5711%	3.5659%	2.0985%	4.7976%
12	2.1347%	2.1053%	1.0611%	4.9124%
13	1.3008%	1.3170%	0.4307%	0.6005%
14	0.8490%	0.8609%	0.2142%	0.3373%
15	0.5431%	0.5515%	0.0935%	0.1890%
16	0.3870%	0.3931%	0.0505%	0.1286%
17	0.1208%	0.1225%	0.0142%	0.0592%
18	0.0759%	0.0770%	0.0110%	0.0283%
19	0.0486%	0.0494%	0.0036%	0.0199%
20	0.0412%	0.0415%	0.0032%	0.0400%
21-30	0.0947%	0.0964%	0.0056%	0.0067%
31-40	0.0053%	0.0055%	0.0012%	0.0000%

8 password policy early on, and only users who have accounts near the beginning have shorter passwords.

Table 3.8 shows the character distribution. It is interesting to note that passwords in American datasets consist of about 27% digits and 69% lower-case letters, while those in Chinese datasets are the opposite, with 68% digits and 30% lower-case letters. This is likely due to both the fact that the Roman alphabet is not native to Chinese, and the fact

Table 3.7.: Password length information for Chinese password datasets

	All Chinese	**CSDN**	**Duduniu**	**178**
4	0.2013%	0.1041%	0.4226%	0.0023%
5	0.3338%	0.5142%	0.5035%	0.0006%
6	11.2856%	1.2954%	9.1683%	20.9249%
7	13.6824%	0.2696%	16.3914%	19.9063%
8	25.8024%	36.3793%	23.0194%	21.6768%
9	21.9962%	24.1479%	24.6739%	17.2318%
10	15.0372%	14.4810%	20.0710%	9.3403%
11	6.1907%	9.7820%	3.4728%	6.9350%
12	2.6199%	5.7475%	1.1962%	2.1269%
13	1.1738%	2.6106%	0.5061%	0.9638%
14	0.8902%	2.4105%	0.3003%	0.5269%
15	0.4598%	1.1719%	0.1895%	0.2822%
16	0.2195%	0.7716%	0.0255%	0.0633%
17	0.0320%	0.1090%	0.0124%	0.0013%
18	0.0287%	0.0918%	0.0107%	0.0058%
19	0.0115%	0.0356%	0.0060%	0.0013%
20	0.0229%	0.0782%	0.0082%	0.0015%
21-30	0.0110%	0.0000%	0.0191%	0.0090%
31-40	0.0012%	0.0000%	0.0029%	0.0000%

that digit sequences are easier to remember in Chinese. For each digit, there are many Chinese characters that have similar sounds, making it easy to find digit sequences that sound similar to some easy-to-remember phrases. Indeed, we found many such sequences occurring frequently in Chinese datasets.[1]

Table 3.9 shows the frequencies of different patterns as well as the most popular passwords for each pattern in all datasets. The frequencies of these patterns in each dataset are listed in Table 3.10. While all lower-case passwords are the most common in American datasets (41.65%), they account for only 8.93% of Chinese datasets. The overall most common pattern is lowercase followed by digits (33.02%), due to the fact that this is the most common in Chinese datasets (40.05%). This is followed by all lower-case, all digits, and digits followed by lower-case; these top-4 patterns account for close to 90% of all passwords. Upper-case letters are most commonly seen preceding a digit sequence, or in all-uppercase passwords. We also note that the pattern distribution shows a lot of variation across different datasets.

Training/Testing Scenarios. We now describe our selection of the training and testing datasets. We decided against merging all datasets into one big dataset and then partitioning it into training and testing, since we feel that represents unrealistic scenarios in practice. For example, this causes similar length, character, and pattern distributions in training and testing; furthermore, any frequent password tends to appear both in training and testing. Instead, in each scenario, we chose some of the datasets for training, and another for testing. Table 3.11 lists the 6 scenarios we use in this chapter. Scenarios 1-4 have training and testing from within the same group (American or Chinese). We merge Yahoo and PhpBB together because they are both small (containing less than one million passwords when combined) when compared with other datasets (all contain more than 6 million). Scenarios 2-4 resemble cross-validation, rotating among the 3 Chinese datasets, each time training with 2 and testing with the remaining 1. Scenario 5 trains on all Chinese datasets and test on the American datasets Yahoo+PhpBB. Scenario 6 trains on Rockyou and tests on the

[1]One such sequence is “5201314”, which sounds similar to a phrase that roughly means “I love you forever and ever”. Both “520” and “1314” are extreme frequent in Chinese datasets.

Table 3.8.: Password characters information

	All	All American	RockYou	PhpBB	Yahoo	All Chinese	CSDN	Duduniu	178
all	490102135	262266576	256756616	1857611	3652349	227835559	60788099	93174301	73873159
digit	46.20%	27.28%	27.35%	23.12%	24.56%	67.99%	67.41%	64.74%	72.55%
lower	51.06%	68.87%	68.78%	73.65%	72.55%	30.55%	30.06%	33.79%	26.87%
special	0.48%	0.67%	0.68%	0.32%	0.49%	0.25%	0.62%	0.12%	0.13%
upper	2.26%	3.18%	3.19%	2.91%	2.39%	1.21%	1.91%	1.35%	0.46%

Table 3.9.: Password pattern information: **L** denotes a lower-case sequence, **D** denotes a digit sequence, **U** denotes a upper-case sequence, and **S** denotes a symbol sequence; patterns differ from templates in that patterns do not record length of sequence.

	ALL		**Percentage of the patterns**	
	Percentage of the pattern	the most popular string	All American	All Chinese
LD	33.22%	a123456	27.79%	40.05%
L	27.15%	password	41.65%	8.93%
D	24.50%	123456	15.77%	35.48%
DL	4.81%	123456aa	2.57%	7.63%
LDL	1.60%	love4ever	1.66%	1.53%
UD	1.48%	A123456	1.33%	1.67%
U	0.94%	PASSWORD	1.48%	0.26%
ULD	0.64%	Password1	0.96%	0.24%
DLD	0.43%	123aa123	0.43%	0.44%
LDLD	0.42%	hi5hi5	0.43%	0.40%
UL	0.40%	Password	0.65%	0.09%
LSD	0.40%	xxx_01	0.66%	0.27%
LSL	0.40%	rock you	0.50%	0.08%
LS	0.38%	iloveyou!	0.64%	0.07%
DU	0.23%	123456A	0.15%	0.34%

Table 3.10.: Percentage of password patterns in datasets: **L** denotes a lower-case sequence, **D** denotes a digit sequence, **U** denotes a upper-case sequence, and **S** denotes a symbol sequence; patterns differ from templates in that patterns do not record length of sequence

	American datasets			**Chinese datasets**		
	Rockyou	Phpbb	Yahoo	Csdn	Duduniu	178
LD	27.71%	19.26%	38.31%	26.15%	55.57%	31.12%
L	41.70%	50.08%	33.05%	11.65%	7.29%	9.00%
D	15.94%	11.94%	5.86%	45.02%	19.48%	48.07%
DL	2.54%	2.05%	5.32%	5.89%	9.79%	6.25%
LDL	1.62%	3.66%	3.31%	1.64%	1.68%	1.27%
UD	1.35%	0.37%	0.56%	1.62%	2.57%	0.62%
U	1.50%	0.73%	0.40%	0.47%	0.21%	0.15%
ULD	0.94%	1.04%	2.48%	0.50%	0.26%	0.05%
DLD	0.42%	0.79%	0.94%	0.52%	0.44%	0.38%
LDLD	0.42%	1.03%	0.97%	0.47%	0.31%	0.47%
UL	0.65%	1.22%	0.70%	0.09%	0.15%	0.01%
LSD	0.66%	0.33%	0.17%	0.21%	0.05%	0.01%
LSL	0.50%	0.17%	0.39%	0.66%	0.21%	0.08%
LS	0.65%	0.16%	0.20%	0.14%	0.05%	0.04%
DU	0.15%	0.11%	0.06%	0.46%	0.46%	0.11%

Chinese dataset CSDN. By comparing scenario 5 against 1, and scenario 6 against 2, one can observe the effect of training on a mismatched dataset. We present detailed results using graphs for scenario 1 and 2, and ANLL$_{0.8}$ for other scenarios.

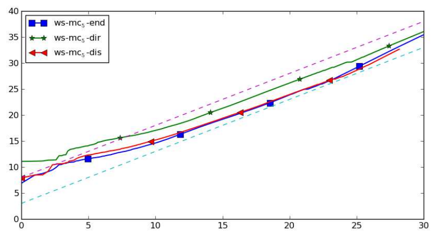
Table 3.11.: Six experimental scenarios.

#	name	Training	Testing
1	Rock→Ya+Ph	Rockyou	Yahoo+PhpBB
2	Du+178→CSDN	Duduniu+178	CSDN
3	CS+178→Dudu	CSDN+178	Duduniu
4	CS+Du→178	CSDN+Duduniu	178
5	Chin→Ya+Ph	Three Chinese	Yahoo+PhpBB
6	Rock→CSDN	Rockyou	CSDN

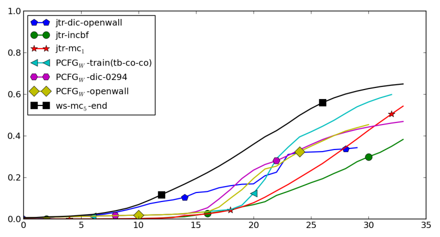
3.5 Experimental Results

Experimental results are presented using guess-number graphs, probability-threshold graphs, and ANLL$_{0.8}$ values. The algorithm naming convention is as follows. Names for whole-string models start with "ws". For example, ws-mc-b$_{10}$ is Markov chain with backoff and frequency threshold 10, and ws-mc$_i$ is order-i Markov chain with add-δ smoothing for $\delta = 0.01$. The postfix -g is for grouping, -ag for grouping after our adaption, -gts for Good-Turing smoothing, and -end, -dir, and -dis are for the three normalization approaches. Names for template-based models start with "tb"; for example, tb-co-mc$_i$ is the template-based model using the counting-based method for assigning probabilities to templates and an order-i Markov chain model for segment instantiation. We note that using this notation, tb-co-co is PCFG$_W$ with dictionary generated from the training dataset.

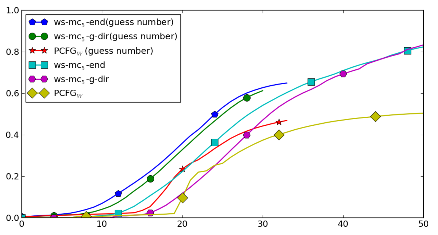
The Figures. Fig 3.2 and Fig 3.3 gives 8 graphs for Scenario 1. Fig 3.2(a) shows the rank vs. probability based on generated passwords. One can see that for the three Markov models shown there, one can translate log of rank into negative log of probability via an additive factor in the range of 3 to 8. That is, a password that has probability $\frac{1}{2^y}$ is likely to have a rank of around 2^{y-a}, which a is mostly between 3 and 8, and seems to gets close to around 6 as x increases. We conjecture that this trend will further hold as x increases,



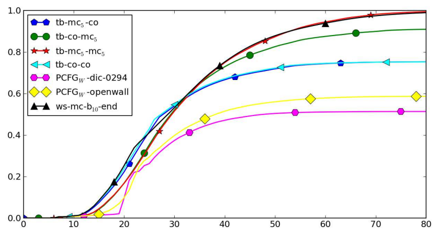
(a) Rank vs. probability: (x, y) denotes the 2^x most likely password has probability $\frac{1}{2^y}$; dashed lines are $y = x + 3$ and $y = x + 8$



(b) Guess-number graph: (x, y) denotes 2^x guesses cover y portion of the dataset



(c) Superimposing guess number and prob. threshold: for guess-number curves, x stands for 2^x guesses; for threshold curves, x stands for probability threshold $\frac{1}{2^x}$; dic-0294 is used as dictionary for PCFG$_W$

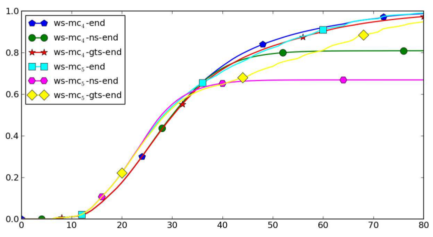


(d) Prob. threshold graph for comparing template-based models (including PCFG$_W$)

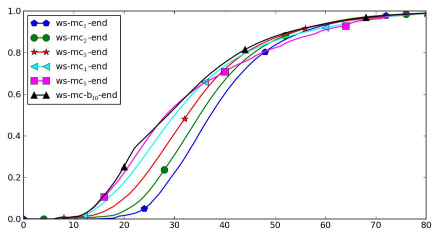
Figure 3.2.: Guess number graphs and probability threshold graphs for Scenario 1: Rock$\rightarrow$Ya+Ph. (Part 1)

and the gap between different curves would shrink as x increases. One support is that these curves have to cross at some point due to the fact that the probabilities for each model add up to 1. Analyzing such Markov models to either prove or disapprove this conjecture is interesting future research, since it affects to what extend one can use probability threshold graphs instead of guess number graphs to compare these models with each other.

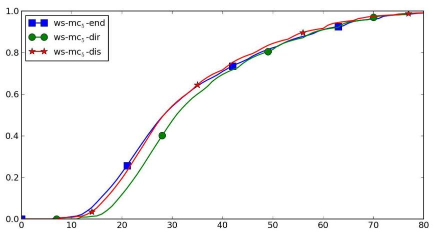
Fig 3.2(b) shows the guess-number graph. We include results from ws-mc$_5$-end, three instantiations of PCFG$_W$ (using dic-0294, the dictionary used in [18], using the Openwall dictionary [61], and using a dictionary generated from the training dataset), and JTR in three different modes (incremental bruteforce, Markov chain, and wordlist/dictionary mode with Openwall dictionary). We can see that the three PCFG$_W$ and the three JTR methods clearly underperform the Markov model. We note that jtr-mc1 seems to pick up rather



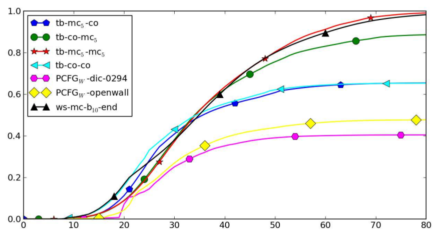
(a) Prob. threshold graph for comparing the effect of smoothing, all with end-based normalization



(b) Prob. threshold graph for comparing Markov of different orders; with end-based normalization and add-δ smoothing



(c) Prob. threshold graph for comparing the effect of normalization



(d) Prob. threshold graph for passwords with length no less than 8; comparing template-based models (including PCFG$_W$)

Figure 3.3.: Guess number graphs and probability threshold graphs for Scenario 1: Rock$\rightarrow$Ya+Ph. (Part 2)

quickly, which matches the shape of ws-mc1 in 3.3(b). Another observation is PCFG$_W$ with training dataset as dictionary (i.e., tb-co-co) outperforms the other two instantiations.

Fig 3.2(c) plots both guess-number curves and probability threshold curves for 3 password models on the same graph, and one can see that the guess-number curve for any model approximately matches the the probability threshold curve if one shifts them to the right.

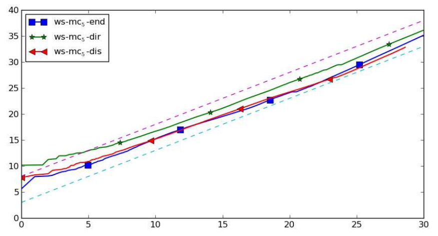
Fig 3.2(d) shows the probability threshold curves for all template-based models and one whole-string model, namely ws-mc-b$_{10}$-end. Here, Laplace smoothing with $\delta = 0.01$ is applied to Markov chains in all template models. PCFG$_W$ with dic-0294 performs the worst, and PCFG$_W$ with Openwall dictionary performs slightly better. Compared with other models, these two cover the least passwords at any probability threshold. With probability threshold at 2^{-80}, the former covers slightly over 50% of all passwords, and the latter covers close to 60%. The two curves that both use counting to instantiate segments (tb-co-co and tb-mc$_5$-co) almost overlap; they perform better than PCFG$_W$ with external dictionaries. On lower probability thresholds, they, together with ws-mc-b$_{10}$, are the best-performing methods At threshold 2^{-80}, they cover around 75% of passwords. This suggests that learning from the dataset is better than using existing dictionaries in PCFG$_W$. When we replace counting with Markov models for instantiating segments, we see another significant improvement at higher probability threshold. The model tb-co-mc$_5$ covers more than 90% of passwords (at threshold 2^{-80}), and tb-mc$_5$-mc$_5$, which takes advantage of smoothing, covers close to 100% passwords. This improvement, however, comes at the cost of slightly worse performance than tb-co-co and tb-mc$_5$-co at lower probability thresholds. In other words, whether to use counting or Markov chains to generate probabilities for templates shows a small difference. Using counting to instantiate segments shows an overfitting effect, performing well at low thresholds, but worse at higher ones. Whole-string Markov with backup (ws-mc-b$_{10}$-end) almost always has the best performance at any threshold.

Fig 3.3(a) compares the effect of no smoothing, add-δ smoothing, and Good-Turing smoothing on Markov models of order 4 and order 5. When $x < 35$, smoothing makes almost no difference, one simply sees that order 5 outperforms order 4. This is to be expected, since the smoothing counts make a difference only for strings of low probabilities. For larger x values, however, smoothing makes a significant difference, and the difference is much more pronounced for order 5 than for order 4. The order 5 model without smoothing performs the worst, due to overfitting. Good-Turing smoothing under-performs add-δ smoothing, and results in significant overfitting for order 5.

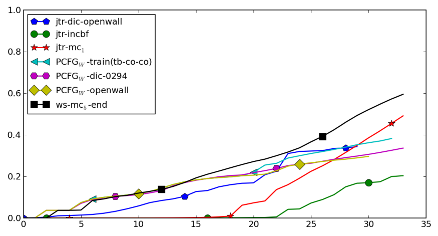
Fig 3.3(b) compares the effect of different orders in Markov chain models. We see that higher-order chains perform better for smaller x values, but are surpassed by lower-order chains later; however, backoff seems to perform well across all ranges of x.

Fig 3.3(c) demonstrates the effect of normalization. Direct normalization performs the worst, while distribution based normalization performs slightly better than end-symbol normalization.

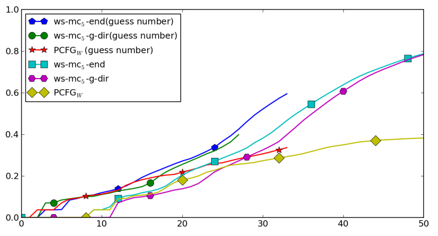
As can be seen from Table 3.6, Yahoo+PhpBB have between 40% and 45% passwords that are of length less than 8. Since many modern websites require passwords to be at least 8 characters long, one may question to what extent results from the above figures are applicable. To answer this question, we repeat figure Fig 3.2(d) by using only passwords that are at least 8 characters for evaluation. The result is shown in Fig 3.3(d). Note that while all curves are somewhat lower than the corresponding ones in Fig 3.2(d); they tell essentially the same story.



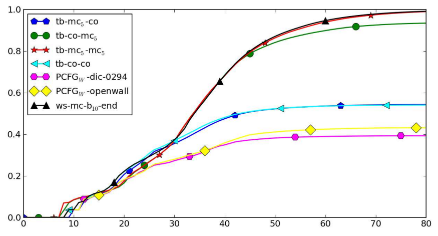
(a) Rank vs. probability: (x, y) denotes the 2^x most likely password has probability $\frac{1}{2^y}$; dashed lines are $y = x + 3$ and $y = x + 8$



(b) Guess-number graph: (x, y) denotes 2^x guesses cover y portion of the dataset



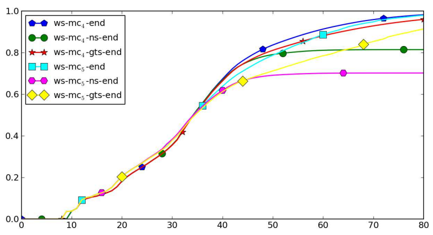
(c) Superimposing guess number and prob. threshold: for cracking curve, x stands for 2^x guesses; for threshold curve, x stands for probability threshold $\frac{1}{2^x}$; dic-0294 is used as dictionary for PCFG$_W$



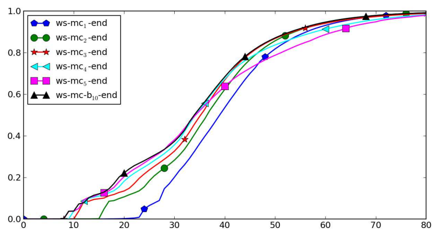
(d) Prob. threshold graph for comparing template-based models (including PCFG$_W$)

Figure 3.4.: Guess number graphs and probability threshold graphs for Scenario 2: Du+178→CSDN. (Part 1)

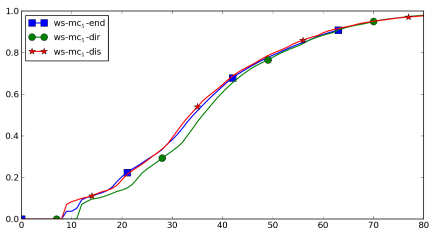
Fig 3.4 and Fig 3.5 gives the same 8 graphs for scenario 2, which use Chinese datasets for training and testing. The observations made above similarly apply. One minor difference is that in Fig 3.2(d), the performance of PCFG with external dictionaries are worse than in Scenario 1. Since the Chinese datasets consist of more passwords that use only digit sequences, and thus are intuitively weaker, this may seem a bit counter-intuitive. This is because PCFG$_W$ uses only digit sequences that appear in the training dataset to instantiate



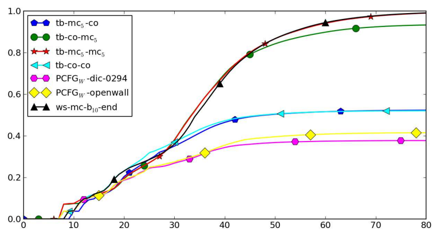
(a) Prob. threshold graph for comparing the effect of smoothing, all with end-based normalization



(b) Prob. threshold graph for comparing Markov of different orders; with end-based normalization and add-δ smoothing



(c) Prob. threshold graph for comparing the effect of normalization

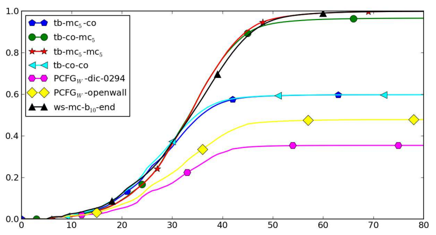


(d) Prob. threshold graph for passwords with length no less than 8; comparing template-based models (including PCFG$_W$)

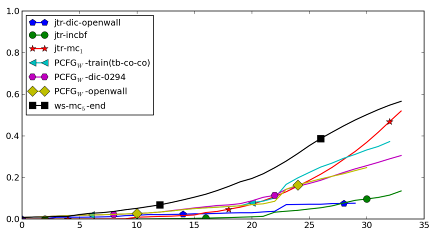
Figure 3.5.: Guess number graphs and probability threshold graphs for Scenario 2: Du+178→CSDN. (Part 2)

password guesses, and thus does perform well when lots of digits are used. When Markov chains with smoothing are used to instantiate the segments, one obtains a more significant improvement than in Scenario 1.

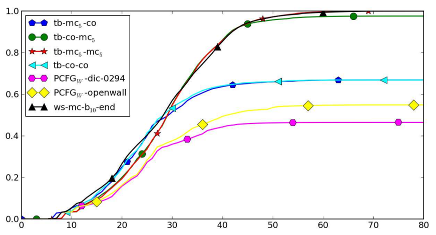
Fig 3.6 and Fig 3.7 shows two of these graphs for each of the other 4 scenarios. These two are the Fig 3.6(d) for a probability threshold graph Fig 3.6(b) for a guess number graph. They mostly give the same observations. We note that in Fig 3.7(b), we see that PCFG$_W$ with dictionary from the training dataset starts out-performing ws-mc$_5$-end. Note that in scenario, we train on the Chinese dataset and the use American datasets to evaluate, thus, a higher-order Markov chain does not perform very well. From Fig 3.7(a), however, we can see that the variable-order ws-mc-b$_{10}$ remains the best-performing method.



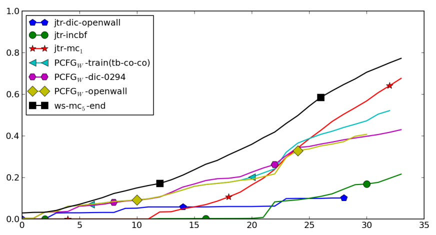
(a) Scenario 3: CS+178→Dudu: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



(b) Scenario 3: CS+178→Dudu: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset



(c) Scenario 4: CS+Du→178: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



(d) Scenario 4: CS+Du→178: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset

Figure 3.6.: Guess number graphs and probability threshold graphs for Scenarios 3, 4.

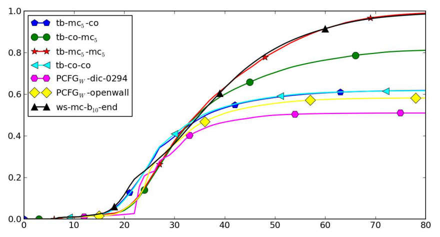
Table 3.12.: ANLL$_{0.8}$ results for scenarios 1, 2, 3. *end*, *dir*, and *dis* stand for end-symbol, direct, and distribution-based normalization, respectively. We highlight the best results within each scenario.

	1: Rock→Ya+Ph			2: Du+178→CSDN			3: CS+178→Dudu		
Algorithm	end	dir	dis	end	dir	dis	end	dir	dis
ws-mc-b$_{10}$	**20.5**	23.3	21.4	**22.7**	25.4	23.4	**23.6**	26.0	23.9
ws-mc-b$_{25}$	**20.8**	23.6	21.7	**22.8**	25.5	23.5	**23.8**	26.1	24.1
ws-mc$_1$	27.7	28.7	26.6	28.7	29.4	27.3	28.0	28.7	26.7
ws-mc$_2$	25.8	27.1	25.0	25.8	26.7	24.7	25.9	27.0	24.9
ws-mc$_3$	23.6	25.2	23.2	24.2	25.3	23.3	24.7	25.9	23.9
ws-mc$_4$	21.7	23.7	21.8	23.3	24.8	**22.8**	24.1	25.6	**23.6**
ws-mc$_5$	**21.1**	23.2	21.3	23.5	25.1	23.2	24.7	26.2	24.3

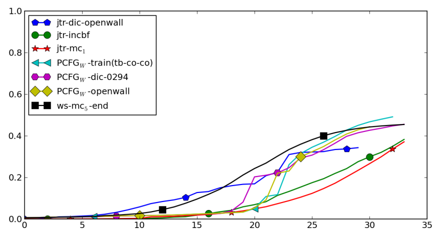
continued on next page

Table 3.12.: *continued*

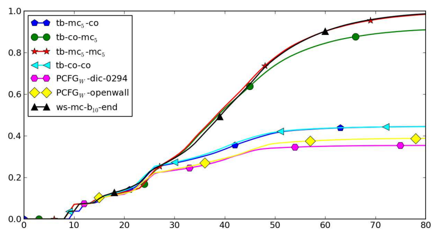
	1: Rock→Ya+Ph			2: Du+178→CSDN			3: CS+178→Dudu		
Algorithm	end	dir	dis	end	dir	dis	end	dir	dis
ws-mc$_1$-g	27.7	28.7	26.7	28.8	29.4	27.4	28.1	28.8	26.8
ws-mc$_2$-g	25.8	27.2	25.1	25.9	26.8	24.8	26.0	27.0	25.0
ws-mc$_3$-g	23.7	25.3	23.3	24.3	25.4	23.4	24.8	26.0	24.0
ws-mc$_4$-g	21.8	23.8	21.8	23.4	24.9	22.9	24.2	25.7	**23.7**
ws-mc$_5$-g	**21.1**	23.2	21.3	23.5	25.1	23.2	24.6	26.2	24.2
ws-mc$_6$-g	21.3	23.3	21.4	25.3	26.8	24.9	25.9	27.4	25.4
ws-mc$_1$-ag	27.6	28.7	26.6	28.7	29.3	27.3	28.0	28.7	26.7
ws-mc$_2$-ag	25.8	27.1	25.0	25.8	26.7	24.7	26.0	27.0	25.0
ws-mc$_3$-ag	23.6	25.3	23.3	24.2	25.3	23.3	24.7	26.0	24.0
ws-mc$_4$-ag	21.7	23.7	21.8	23.3	24.8	**22.8**	24.1	25.6	**23.7**
ws-mc$_5$-ag	**21.0**	23.1	21.2	23.4	25.0	23.1	24.6	26.1	24.2
ws-mc$_6$-ag	21.3	23.3	21.3	25.1	26.7	24.7	25.9	27.3	25.3
ws-mc$_1$-gts	27.7	28.7	26.6	28.7	29.4	27.3	28.0	28.7	26.7
ws-mc$_2$-gts	25.8	27.1	25.1	25.8	26.7	24.7	25.9	27.0	24.9
ws-mc$_2$-gts	23.6	25.3	23.3	24.2	25.3	23.3	24.7	26.0	24.0
ws-mc$_2$-gts	22.0	23.9	22.0	23.5	24.9	23.0	24.3	25.8	23.9
ws-mc$_2$-gts	22.5	24.4	22.5	24.7	26.1	24.3	26.8	28.0	26.1
tb-mc$_1$-mc$_1$	27.8	28.8	26.7	28.8	29.4	27.3	28.0	28.8	26.7
tb-mc$_2$-mc$_2$	26.2	27.3	25.3	26.2	26.9	24.9	26.4	27.3	25.3
tb-mc$_3$-mc$_3$	24.4	25.7	23.7	24.7	25.6	23.6	25.3	26.4	24.4
tb-mc$_4$-mc$_4$	22.8	24.3	22.3	23.9	25.0	23.0	24.7	26.0	24.0
tb-mc$_5$-mc$_5$	21.9	23.6	21.6	23.5	24.8	**22.8**	24.4	25.9	23.9



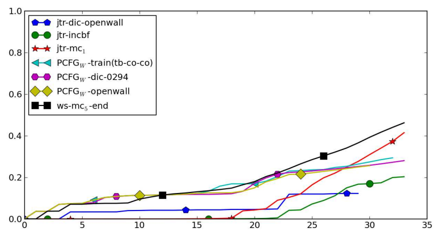
(a) Scenario 5: Chin→Ya+Ph: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



(b) Scenario 5: Chin→Ya+Ph: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset



(c) Scenario 6: Rock→CSDN: Prob. threshold graph for comparing template-based models (including PCFG$_W$)



(d) Scenario 6: Rock→CSDN: Guess-number graph. (x, y) denotes 2^x guesses cover y portion of the dataset

Figure 3.7.: Guess number graphs and probability threshold graphs for Scenarios 5, 6.

Table 3.13.: ANLL$_{0.8}$ results for scenarios 4, 5, 6. end, dir, and dis stand for end-symbol, direct, and distribution-based normalization, respectively. We highlight the best results within each scenario.

	4: CS+Du→178			5: Chin→Ya+Ph			6: Rock→CSDN		
Algorithm	end	dir	dis	end	dir	dis	end	dir	dis
ws-mc-b$_{10}$	**19.3**	22.0	20.2	**24.8**	26.9	**25.1**	**26.4**	28.9	27.2
ws-mc-b$_{25}$	**19.4**	22.1	20.3	**25.4**	27.4	25.6	**26.6**	29.1	27.4
ws-mc$_1$	24.8	25.6	23.7	30.9	31.7	29.9	31.1	31.3	29.6
ws-mc$_2$	22.3	23.5	21.7	29.6	30.5	28.7	29.0	29.5	27.7
ws-mc$_3$	20.8	22.3	20.5	27.2	28.4	26.6	27.8	28.5	26.7
ws-mc$_4$	19.9	21.8	20.0	25.9	27.4	25.7	27.4	28.2	**26.6**
ws-mc$_5$	**19.5**	21.6	19.8	26.4	27.8	26.0	27.9	29.0	27.3

continued on next page

Table 3.13.: *continued*

	4: CS+Du→178			5: Chin→Ya+Ph			6: Rock→CSDN		
Algorithm	end	dir	dis	end	dir	dis	end	dir	dis
ws-mc$_1$-g	24.8	25.6	23.8	30.9	31.7	29.9	31.1	31.4	29.6
ws-mc$_2$-g	22.4	23.5	21.7	29.6	30.6	28.7	29.0	29.6	27.8
ws-mc$_3$-g	20.9	22.4	20.6	27.2	28.5	26.7	27.9	28.5	26.8
ws-mc$_4$-g	20.0	21.8	20.0	25.8	27.4	25.6	27.4	28.2	**26.6**
ws-mc$_5$-g	**19.5**	21.6	19.8	25.9	27.4	25.6	27.7	28.8	27.1
ws-mc$_6$-g	20.2	22.3	20.5	26.3	27.5	25.7	30.0	31.2	29.4
ws-mc$_1$-ag	24.8	25.6	23.7	30.9	31.7	29.8	31.0	31.3	29.5
ws-mc$_2$-ag	22.3	23.5	21.6	29.6	30.5	28.7	28.9	29.4	27.7
ws-mc$_3$-ag	20.8	22.3	20.5	27.2	28.4	26.6	27.8	28.4	26.7
ws-mc$_4$-ag	19.9	21.8	20.0	25.8	27.4	25.6	27.3	28.1	**26.5**
ws-mc$_5$-ag	**19.4**	21.6	19.8	25.8	27.3	**25.5**	27.5	28.7	27.0
ws-mc$_6$-ag	20.0	22.2	20.5	26.2	27.5	25.7	29.9	31.0	29.3
ws-mc$_1$-gts	24.8	25.6	23.7	30.9	31.7	29.9	31.1	31.3	29.6
ws-mc$_2$-gts	22.3	23.5	21.7	29.6	30.5	28.7	29.0	29.5	27.7
ws-mc$_2$-gts	20.8	22.4	20.5	27.3	28.5	26.7	27.8	28.5	26.8
ws-mc$_2$-gts	20.0	21.8	20.0	26.9	28.3	26.6	27.8	28.6	27.0
ws-mc$_2$-gts	19.6	21.7	19.9	30.2	31.0	29.2	30.4	31.3	29.6
tb-mc$_1$-mc$_1$	24.9	25.6	23.7	30.9	31.7	29.9	31.0	31.3	29.5
tb-mc$_2$-mc$_2$	22.8	23.8	22.0	29.8	30.7	28.9	29.1	29.5	27.8
tb-mc$_3$-mc$_3$	21.6	22.8	21.0	27.8	28.8	27.0	27.9	28.5	26.8
tb-mc$_4$-mc$_4$	20.9	22.4	20.5	26.4	27.6	25.8	27.1	27.9	**26.2**
tb-mc$_5$-mc$_5$	20.4	22.1	20.3	25.8	27.2	**25.4**	26.8	27.8	**26.1**

ANLL Table. ANLL$_{0.8}$ values for all six scenarios are given in Table 3.12 and Table 3.13. In the tables, end, dir, and dis stand for end-symbol, direct, and distribution-based normalization, respectively. We highlight the best results within each scenario. Using this format, we can compare more models directly against each other, with the limitation that these results need to be interpreted carefully. Some models assign probability 0 to some passwords; their ANLLs are not well-defined and thus not included. Results for those models are presented using graphs.

Many observations can be made from the tables. First, backoff with end-symbol normalization is gives the best result overall, as it produces results that are among the best across all scenarios. Especially, for Scenario 1, which we consider to be the most important one, it produces the best overall result. Several other models perform quite close. It seems that using a Markov chain of an order that is high enough, but not too high, and with some ways to deal with overfitting, would perform reasonably well.

Second, for most other models, distribution-based normalization performs the best, followed by end-symbol normalization. Direct normalization, which was implicitly used in the literature, performs the worst. Yet, for backoff, end-symbol normalization performs the best. There seems to exist some synergy between backoff and end-symbol normalization. One possible reason is that as backoff uses variable-length Markov chains, it can recognize long common postfixes of passwords so that it can end a password appropriately, instead of depending only on the length of passwords for normalization. With fixed-length Markov chains, one does not have this benefit.

Third, on the effect of smoothing, Good-Turing smoothing performs unexpectedly poorly, especially for higher-order Markov chains; it seems that they tend to cause more overfitting, a phenomenon also shown in Figure 3.3(a) and 3.5(a). For higher orders Markov models, add-δ smoothing, grouping, adapted grouping, and template-based models all perform similarly; they are just slightly worse than backoff with end-symbol normalization.

Fourth, for most models, the Markov chain order that gives the best results varies from scenario to scenario. For Scenario 1, order-5 appears to be the best. Yet for the scenarios with Chinese datasets (2, 3), order 4 generally outperform order 5. One obvious reason is the slightly smaller training dataset. Also, because the Chinese datasets use digits much more than letters, they contain even non-digit sequences for training, resulting in better performance for lower-order Markov chains. Again, this demonstrates the benefit of using a variable-order model like backoff, since one does not need to choose the appropriate order.

Fifth, comparing the pair of scenarios 5 and 1, and the pair of 6 and 2, one can see a difference in $\text{ANLL}_{0.8}$ of about 2 to 4 in each case; this demonstrates the importance of training on a similar dataset.

Sixth, comparing scenarios 2, 3, and 4, we can see that 178 is clearly the weakest password dataset among the 3, which is corroborated by evidence from Table 3.5 and Table 3.9. In 178, 55% of passwords are those appearing more than 5 times (compared to 30% and 25% for CSDN and Duduniu); close to 21% are length 6 (compared to 1.3% and 9%); 48% are all digits (compared to 45% for CSDN and 19.5% for Duduniu; recall that passwords in CSDN tend to be significantly longer).

3.6 Conclusions

In summary, we make the following contributions in this chapter:

- We introduce the methodology of using probability-threshold graphs for password research, which has clear advantages over the current standard approach of using guess-number graphs for type-1 password research. They are much faster to construct and also gives information about the passwords that are difficult to crack. In our experiments, it took about 24 hours to generate 10^{10} passwords for plotting guess-number graphs; which cover between 30% and 70% in the dataset. On the other hand, it took less than 15 minutes to compute probabilities for all passwords in a size 10^7 testing dataset, giving strength information of all passwords in the dataset. We note, however, that for type-2 research, in which one compares different password models, one needs to be careful to interpret results from probability-threshold graphs and should use guess-number graphs to corroborate these results.

- We introduce knowledge and techniques from the rich literature in n-gram models for statistical language modeling into password modeling, as well as identifying new issues that arise from modeling passwords. We also identify a broad design space for password models.

- We conduct a systematic evaluation of many password models, and make a number of findings. In particular, we show that the PCFG$_W$ model, which has been assumed to be the state of the art and is widely used in research, does not perform as well as whole-string Markov models.

4. IMPROVING PASSWORD POLICIES AND PRACTICE USED BY WEBSITES

In Chapter 3, we have introduced metrics comparing probabilistic password models. Given a model that can accurately measure the strength of passwords, a natural question raised is how to apply the model in practice. In particular, how can the model be used to help users choose strong passwords.

In this chapter, we address the question: How to best check weak passwords? We model different password strength checking methods as Password Ranking Algorithms (PRAs), and introduce two methods for comparing different PRAs: the β-Residual Strength Graph (β-RSG) and the Normalized β-Residual Strength Graph (β-NRSG). In our experiments, we find some password datasets that have been widely used in password research contain many problematic passwords that are not naturally created. We develop techniques to cleanse password datasets by removing these problematic accounts. We then apply the two metrics on cleansed datasets and show that several PRAs, including the dictionary-based PRA, the Markov Models with and without backoff, have similar performances. If the size of PRAs are limited in order to be able to be transmitted over the internet, a hybrid method combining a small dictionary of weak passwords and a Markov model with backoff with a limited size can provide the most accurate strength measurement.

4.1 Introduction

Password-based authentication is the most widely used authentication mechanism. Despite countless attempts at designing mechanisms to replace it, password-based authentication appears more widely used and firmly entrenched than ever [1, 28, 62]. One major weakness of password-based authentication is the inherent tension between the security and usability of passwords [2, 3]. More precisely, secure passwords tend to be difficult to memorize (i.e., less usable) whereas passwords that are memorable tend to be predictable. Generally individuals side with usability of passwords by choosing predictable and weak passwords [2, 4–7].

To deal with this, the most common approach is to forbid the use of weak passwords, or give warnings for passwords that are "somewhat weak". This approach requires an effective way to identify weak passwords. One way is to use password composition policies, i.e., requiring passwords to satisfy some syntactical properties, e.g., minimum length and/or categories of characters. An alternative is to use proactive password checkers that are based on a weak password dictionary [32–34]. More recently, probabilistic password models, which work by assigning a probability to each password, were introduced [17–19].

How to best check weak passwords is still an open question. A study in 2014 [21] examined several password meters in use at popular websites and found highly inconsistent strength estimates for the same passwords using different meters. The report did not answer the question of which meter is the best, nor what methods should be used to compare them. Designing an effective password meter requires solving two problems: (1) How to accurately assess the strength of passwords chosen by the users; and (2) How to communicate the strength information to and interact with the users to encourage them to choose strong passwords. These two problems are largely orthogonal. In this chapter we focus on solving the first problem.

We model different password strength assessing methods (including composition policies) as Password Ranking Algorithms (PRAs), which assign a rank to every password. One state-of-the-art method for comparing PRAs is the Guess Number Graph (GNG), which plots the number of guesses vs. the percentage of passwords cracked in the test dataset. However, GNG measures only the total density of the uncracked passwords, but not their distribution, which is critical in assessing the effectiveness to defend against guessing attacks after deploying the PRA. To address this limitation of GNG, we propose the β-Residual Strength Graph (β-RSG), which measures the strength of the β most common passwords in the test dataset, after forbidding the weakest passwords identified by a PRA. When a PRA forbids a large number of passwords that users are extremely unlikely to use, it performs poorly under β-RSG. To limit the influence of these passwords, we also propose Normalized β-Residual Strength Graph (β-NRSG), which ignores how passwords that do not appear in the testing dataset are ranked. β-NRSG also has the advantage that we can use it to evaluate blackbox password strength services for which one can query the strength of specific passwords, but cannot obtain all weak passwords.

Surprisingly, we observe that all PRAs perform significantly worse on password datasets from Chinese websites than on datasets from English websites, because some of the most frequent passwords in the testing dataset are not recognized as weak passwords by all the PRAs. Further investigation reveal that these passwords are in all likelihood due to “fake accounts”, possibly created by site administrators to artificially boost the number of registered users. The evidence for this includes that the user IDs associated with such passwords look suspicious. These suspicious IDs fall into two categories: appending a counter to a fixed prefix; and a large number of fixed-length strings that apparently look random. While these datasets have been used in previous papers, we are the first to report such fake accounts. We develop a data cleansing technique to identify and remove such “fake” accounts in order to obtain a more accurate evaluation.

Our evaluation is based on the cleansed password datasets. We have compared the Probabilistic Context-Free Grammar (PCFG) [18] method, Markov models with and without backoff [17, 19], blacklists based on training datasets, the combined method proposed by Ur et al. [22], password composition policies, as well as two versions of *zxcvbn* [23]. We also show how GNG, β-RSG, and β-NRSG differ. We find that when one places no limit on the mode size, several methods including the blacklist approach, Markov Models, and the Combined method have similar performance. When one wants to check the strength of passwords on the client side, without sending passwords over the network, the model size must be limited. We find that a blacklist with a limited size still provide the most accurate strength measurement for the most popular passwords. However, only a limited number of passwords are covered.

We then propose a new client-end PRA that uses a hybrid method; it uses a small blacklist to assess the strength of most popular passwords, and evaluate the other passwords based on a limited size Markov model with backoff. We show that the hybrid method inherits the advantages of both methods, and consistently outperform the other client-end PRAs.

4.2 How to Compare PRAs

In this section, we propose metrics evaluating password ranking algorithms (PRAs).

At the core of any password strength meter is a Password Ranking Algorithm (PRA), which is a function that sorts passwords from weak (common) to strong (rare).

Definition 4.2.1 (Password Ranking Algorithm (PRA)) *Let $\mathcal{P}$ denote the set of all allowed passwords. A Password Ranking Algorithm $r : \mathcal{P} \to \mathsf{Rnk}$ is a function that maps each password to a ranking in* Rnk*, where* $\mathsf{Rnk} = \{1, \ldots, |\mathcal{P}|\} \cup \{\infty\}$.

Intuitively, a password with rank 1 means that it is considered to be one of the weakest password(s); and a password with rank ∞ means that it is considered to be strong enough to not need a ranking. The above definition accommodates PRAs that rank only a subset of all passwords as well as PRAs that rank some passwords to be of equal strength. A password composition policy can be modeled as a PRA that assigns a rank of 1 to passwords that do not satisfy the policy, and ∞ otherwise. Probabilistic password models that assign a probability to each password can be converted into a PRA by sorting, in decreasing order, the passwords based on their probabilities in the model. Arguably, this captures the essential information for determining the strengths of passwords, since both cracking passwords and choosing which passwords to forbid should be done based on the ranking.

4.2.1 Guess Number Graph (GNG)

The state-of-the-art method for comparing PRAs is the Guess Number Graph (GNG), which plots the number of guesses vs. the percentage of passwords cracked in the dataset. A point (x, y) on a curve means that y percent of passwords are included in the first x guesses. When evaluating PRAs for their effectiveness in cracking passwords, GNG is an ideal choice. For the same x value, a PRA that has a higher y value is better. However, one limitation of GNG is that it does not convey information regarding the distribution of uncracked passwords. For example, suppose that two PRAs r_1 and r_2 both cover 40% of passwords after making 10^6 guesses. Under r_1 there remain uncovered 5 passwords each appearing 200 times and a large number of passwords that appear just once. And under r_2 there remain 500 passwords each appearing 2 times together with a similarly large number of passwords that appear just once. In this case, if we decide to forbid the first 10^6 passwords that are considered weak and an adversary is limited to 5 guess attempts per account (e.g., because of rate limiting), the adversary can successfully break into 1,000 accounts based on r_1, but only 10 accounts based on r_2. Obviously, r_1 is worse than r_2, even though they look the same under the GNG. Therefore, GNG is not appropriate for the effectiveness of using a PRA for identifying and forbidding the usage of weak passwords, especially since the primary objective of checking password strength is to defend against online guessing attacks, as offline attacks are best defended against by improving site security and by using salted, slow cryptographic hash functions when storing password hashes.

4.2.2 The β-Residual Strength Graph (β-RSG)

To deal with the limitation of GNG, we propose to use the β-Residual Strength Graph. Each PRA r corresponds to a curve, such that a point (x, y) on the curve means that after forbidding what r considers to be the x weakest passwords, the strength of the remaining passwords is y. For the choice of y, we use the effective key-length metric corresponding to the β-success-rate, proposed by Boztaş [29] and Bonneau [28], to measure the strength of the probabilities of the remaining passwords, which we call the residual distribution.

More specifically, given a password dataset D, we use $p_D(w)$ to denote a password w's frequency in D, i.e., $p_D(w) = \frac{\text{number of times } w \text{ occurs in } D}{|D|}$. Given a PRA r and a number x, let w_i be the ith most frequent password in D that is not among the x weakest passwords according to r. Then the β-Residual Strength is computed as:

$$y = \lg \left(\frac{\beta}{\sum_{i=1}^{\beta} p_D(w_i)} \right),$$

Intuitively, β-RSG provides a measure of the strength of the remaining weakest passwords after a certain number of weak passwords according to r are forbidden. It translates the total frequencies of the β unremoved weakest passwords into a bit-based security metric, which can be viewed as finding the entropy of a uniform distribution where the probability of each element equals that of the average of these β passwords.

We need to choose appropriate values for β. In [28], $\tilde{\lambda}_{10}$ is used, which corresponds to an online attack setting where 10 guesses are allowed, which was recommended by usability studies [30]. We adapt the setting.

4.2.3 The Normalized β-Residual Strength Graph (β-NRSG)

Password composition policies (such as the ones that require mixing letters with digits and special symbols), when viewed as PRAs, tend to perform poorly under the RSG, because they rule out a large number of passwords, e.g., all passwords that consist of only letters. This observation demonstrates that one weakness of password composition policies is that they prevent some strong passwords (such as unpredictable passwords consisting of only letters) from being used. However, one may argue that this criticism is not completely fair to them. The cost of forbidding a strong (i.e., rarely used) password is that users who naturally want to use such a password cannot do so, and have to choose a different password, which they may have more trouble remembering. However, if users are extremely unlikely to choose the password anyway, then there is very little cost to forbid it.

We thus propose a variation of RSG, which "normalizes" a RSG curve by considering only passwords that actually appear in the testing dataset D. More specifically, a point (x, y) on the curve for a PRA r means that after choosing a threshold such that x passwords that appear in D are forbidden, the residual strength is y. We call this variation the Normalized β-Residual Strength Graph (β-NRSG). A NRSG curve can be obtained from a corresponding RSG curve by shrinking the x axis; however, different PRAs may have different shrinking effects, depending on how many passwords that are considered weak by the PRAs do not appear in the testing dataset. Under β-NRSG, PRAs are not penalized for rejecting passwords that do not appear in the testing dataset. A PRA would perform well if it considers the weak (i.e., frequent) passwords in the dataset to be weaker than the passwords that appear very few times in it. β-NRSG also has the advantage that we can use it to evaluate blackbox password strength services for which one can query the strength of specific passwords, but cannot obtain all weak passwords. We suggest using both RSGs and NRSGs when comparing PRAs.

4.2.4 Client versus Server PRAs

A PRA can be deployed either at the server end, where a password is sent to a server and has its strength checked, or at the client end, where the strength checking is written in JavaScript and executed in the client side inside a browser. PRAs deployed at the server end are less limited by the size of the model. On the other hand, deploying PRAs on the client side increases confidence in using them, especially when password strength checking tools are provided by a third party. Thus it is also of interest to compare the PRAs that have a relatively small model size, and therefore can be deployed at the client end. We say a PRA is a Client-end PRA if the model size is less than 1MB, and a Server-end PRA otherwise.

4.2.5 PRAs We Consider

The PRAs that are considered in this chapter are listed in Table 4.1. In Client-end PRAs, the size of *zxcvbn*$_1$, *zxcvbn*$_2$ are 698KB and 821KB correspondingly. For password models

Table 4.1.: Server-end PRAs and Client-end PRAs. X_c means reduced-size version of model X in order to be deployed at the client side.

Server-end	Markov Model [19], Markov Model with backoff, Probabilistic Context-free Grammar [18], Google API, Blacklist, Combined [22]
Client-end	*zxcvbn*$_1$ [23], *zxcvbn*$_2$ [23], Blacklist$_c$, Markov Model$_c$, Markov Model with backoff$_c$, Hybrid

whose model sizes are adjustable, we make the model size to be approximately 800KB to have a fair comparison.

PCFG. In the PCFG approach [18], one divides a password into several segments by grouping consecutive characters of the same category (e.g., letters, digits, special symbols) into one segment. Each password thus follows a pattern, for example, L_7D_3 denotes a password consisting of a sequence of 7 letters followed by 3 digits. The distribution of different patterns as well as the distribution of digits and symbols are learned from a training dataset. PCFG chooses words to instantiate segments consisting of letters from a dictionary where all words in the dictionary are assumed to be of the same probability. The probability of a password is calculated by multiplying the probability of the pattern by the probabilities of the particular ways the segments are instantiated.

Markov model. N-gram models, i.e., Markov chains, have been applied to passwords [19]. A Markov chain of order d, where d is a positive integer, is a process that satisfies

$$P(x_i|x_{i-1}, x_{i-2}, \ldots, x_1) = P(x_i|x_{i-1}, \ldots, x_{i-d})$$

where d is finite and $x_1, x_2, x_3, \ldots$ is a sequence of random variables. A Markov chain with order d corresponds to an n-gram model with $n = d + 1$.

We evaluate 5-gram Markov Model (MC$_5$), as recommended in Chapter 3, within Server-end PRAs setting. In order to fit the Markov Model into a Client-end PRA, if we store the frequency of each sequence in a trie structure, the leaf level contains 95^n nodes, where 95 is the total number of printable characters. To limit the size of Markov model to be no larger than 1MB, n should be less than 4. We use 3-order Markov Model MC_3 in our evaluation.

Markov model with backoff. The model was proposed in Chapter 3. The intuition of the model is that if a history appears frequently, then we would want to use that to estimate the probability of the next character. In this model, one chooses a threshold and stores all substrings whose counts are above the threshold, and use the frequency of these substrings to compute the probability. Therefore, the model size of a Markov Model with backoff depends on the frequency threshold selected. In this chapter, we consider two sizes of Markov Model with backoff by varying frequency threshold. We first pick a relatively small threshold 25 (MCB_{25}), as suggested in Chapter 3, to construct a Server-end PRA.

Table 4.2.: Model size of Markov models with backoff using different frequency threshold.

Train	Frequency Threshold					
	25	200	500	1000	1500	2000
RockYou	18M	3.4M	1.7M	1M	712K	556K
Duduniu	7.8M	1.5M	604K	368K	268K	200K

For Client-end PRAs, similar to the Markov model, we record the model in a trie structure, where each node contains a character and the corresponding count of the sequence starting from the root node to the current node. We measure the size of data after serializing the trie into JSON format. Table 4.2 shows the size of the models trained on *Rockyou* and *Duduniu* dataset with different frequency thresholds. The size of the Markov Models with backoff when trained on *Duduniu* dataset is significantly smaller than that of models trained on the *Rockyou* dataset. This is primarily due to the difference in character distribution between English and Chinese users. English users are more likely to use letters while Chinese users are more likely to use digits. As a result, the most frequent sequences in *Rockyou* are mainly constructed by letters while those in *Duduniu* are mainly constructed by digits. The difference in the size of the models comes from the different search space in letters and digits. In order to approximate the size of the model to that of *zxcvbn*, we choose MCB_{1500} for English datasets and MCB_{500} for Chinese datasets.

Dictionary-based Blacklist. Dictionary-based blacklists for filtering weak passwords have been studied for decades, e.g., [32–34]. Some popular websites, such as Pinterest and Twitter, embed small weak password dictionaries, consisting of 13 and 401 passwords respectively, on their registration pages. We use a training dataset to generate the blacklist dictionary. The order of the passwords follows the frequency of passwords in the training dataset in a reversed order. Assuming each password contains 8 characters on average, a dictionary with 100,000 passwords is approximately 900KB. Such blacklist (Blacklist$_c$) is used in Client-end PRAs settings.

Combined Method. Ur et al. [22] proposed Min_{auto} metric, which is the minimum guess number for a given password across multiple automated cracking approaches. We implement a password generator which outputs passwords in the order of their corresponding Min_{auto}. Passwords with smaller Min_{auto} are generated earlier. In the Combined PRA, the rank of a password is the order of the passwords generated. In this chapter, Min_{auto} is calculated by combining 4 well-studied approaches: Blacklist, PCFG, Markov, and Markov with backoff.

Google Password Strength API. Google measures the strength of passwords by assigning an integer score ranging from 1 to 4 when registering on their website. We find that the score is queried via an AJAX call and the API is publicly available[1]. We use this service to assess the strength of passwords. We are not able to generate passwords and get the exact ranking as the underlying algorithm has not been revealed.

Zxcvbn Version 1 Zxcvbn is an open-source password strength meter developed by Wheeler [23]. It decomposes a given password into chunks, and then assigns each chunk an estimated "entropy". The entropy of each chunk is estimated depending on the pattern of the chunk. The candidate patterns are "dictionary", "sequence", "spatial", "year", "date", "repeat" and "bruteforce". For example, if a chunk is within the pattern "dictionary", the

[1]https://accounts.google.com/RatePassword

entropy is estimated as the log of the rank of word in the dictionary. Additional entropy is added if uppercase letters are used or some letters are converted into digits or sequences (e.g. a$\Rightarrow$@). There are 5 embedded frequency-ordered dictionaries: 7140 passwords from the Top 10000 password dictionary; and three dictionaries for common names from the 2000 US Census. After chunking, a password's entropy is calculated as the sum of its constituent chunks' entropy estimates.

$$\mathsf{entropy}(pwd) = \sum \mathsf{entropy}(chunk_i)$$

A password may be divided into chunks in different ways, Zxcvbn finds the way that yields the minimal entropy and uses that.

Zxcvbn Version 2 In October 2015, a new version of *zxcvbn* was published. Zxcvbn$_2$ also divides a password into chunks, and computes a password's strength as the "minimal guess" of it under any way of dividing it into chunks. A password's "guess" after being divided into chunks under a specific way is:

$$l! \times \prod_{i=1}^{l} (\mathit{chunk}_i.\mathit{guesses}) + 10000^{l-1}$$

where l is the number of the chunks. The factorial term is the number of ways to order l patterns. The $10000^{(l-1)}$ term captures the intuition that a password that has more chunks are considered stronger. Another change in the new version is that if a password is decomposed into multiple chunks, the estimated guess number for each chunk is the larger one between the chunks' original estimated guess number and a min_guess_number, which is 10 if the chunk contains only one character or 50 otherwise. While these changes are heuristic, our experimental results show these changes cause significant improvements under our methods of comparison.

There are some other subtle changes, such as one built-in dictionary is replaced. For consistency, in this chapter, we use the original dictionaries in both two versions.

Table 4.3.: Examples of differences in two versions of *zxcvbn*.

Password	*zxcvbn*$_1$	*zxcvbn*$_2$
`password1`	2.0	8.8
`passwordsmith`	0.0	13.8

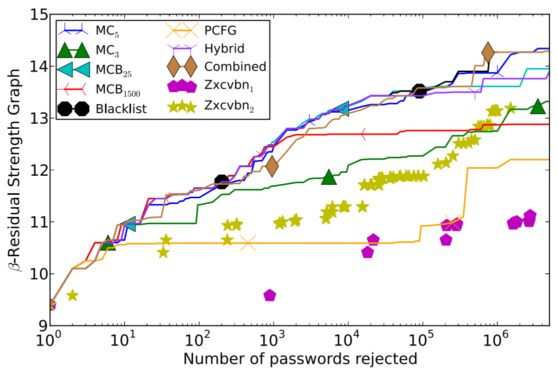
Table 4.3 illustrates the penalty in entropy estimation for passwords which consist of more than one chunk. The estimated entropy for such passwords significantly increase in the second version of *zxcvbn*. Actually, `password1` is decomposed as `password` and `1` in *zxcvbn*$_1$ while viewed as only one chunk in *zxcvbn*$_2$. If decomposed into 2 chunks in *zxcvbn*$_2$, the password's estimated entropy is even larger.

Hybrid Method Observing the promising performance of dictionary methods and the limited number of passwords covered (see Chapter 4.4.2 for details), we propose a hybrid PRA which combines a blacklist PRA with a backoff model. In the hybrid PRA, we reject passwords belonging to a blacklist dictionary or with low scores using the backoff model. To make the size of the PRAs consistent, we further limit the size for both dictionary and backoff model. We chose to use a dictionary containing 30 000 words, which takes less than 300KB. In order to keep the total size of the model consistent, we used MCB_{2000} and MCB_{1000} for English datasets and Chinese datasets, respectively.

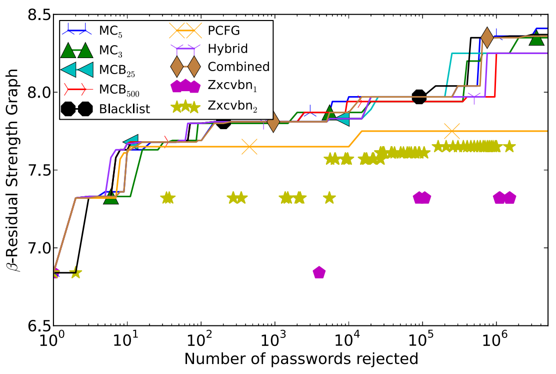
4.3 Data Cleansing

Poor Performance of PRAs on Chinese Datasets. In our evaluation comparing PRAs, we observe that almost all PRAs perform poorly on some Chinese dataset.

Figure 4.1 shows the results of an β-Residual Strength Graph(β-RSG) evaluation on *Xato* (an English dataset) and *178* (a Chinese dataset). A point (x, y) on a curve means if we want to reject the top x passwords from a PRA, the residual strength is y. It is clear that the residual strength for *178* is much lower than that of *Xato*. In *178*, even if 1 million passwords are rejected, the residual strength is around or lower than 8 for all PRAs we examined, which means the average of the remaining top 10 passwords' probability is as



(a) *Xato*, $\beta = 10$



(b) *178*, $\beta = 10$

Figure 4.1.: β-Residual Strength Graph(β-RSG) on original *Xato* and *178* datasets. A point (x, y) on a curve means if we want to reject the top x passwords from a PRA, the strength of the remaining passwords is y.

high as $\frac{1}{2^8} \approx 0.39\%$. We find that 12 out of the top 20 passwords in *178* are not among the first million weakest passwords for any PRA. This led us to investigate why this occurs.

Evidence of Suspicious IDs. We find that the dataset contains a lot of suspicious account IDs which mostly fall in to two patterns: (1) *Counter*: a common prefix appended by a counter; (2) *Random*: a random string with a fixed length. Table 4.4 lists some suspicious accounts sampled from the *178* dataset, which we believe were created either by a single user in order to benefit from the bonus for new accounts, or by the system administrator, in order to artificially boost the number users on the sites. Either way, such passwords are not representative of actual password choices and should be removed.

Table 4.4.: Examples of problematic IDs in *178* dataset.

Password	*Counter* IDs (sampled)	*Random* Ids (sampled)
`zz12369`	badu1; badu2; ...; badu50	vetfg34t; gf8hgoid; vkjjhb49; 5t893yt8; 9y4tjreo; 09rtoivj; kdznjvhb
`qiulaobai`	qiujie0001; qiujie0002; ...; qiujie0345	j3s1b901; ul2c6shx; a3bft0b8; wzjcxytp; 7fmjwzg2; 0ypvjqvo
`123456`	1180ma1; 1180ma2; ...; 1180ma49	x2e03w5suedtu; 7kjwd-dqujornc; inrrgjhm2dh8r; 3u2lnalg91u9i;

Table 4.5.: Number of accounts removed after identifying problematic IDs.

Dataset	*Yahoo*	*Xato*	*Duduniu*	*CSDN*	*178*
Removed	232	9577	9796	69317	1639868
Total	434131	9148094	7304316	6367411	8434340

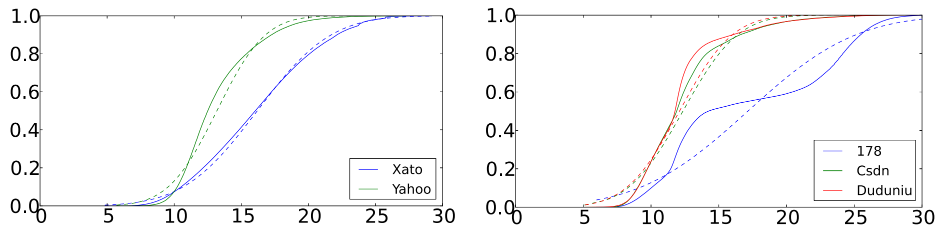
Suspicious Account Detection We detect and remove suspicious passwords (accounts) using the user IDs and email addresses. *Yahoo* and *Duduniu* datasets only have email address available. We first remove the email provider, i.e., the postfix starting from @, and then, treat the prefix of email addresses as account IDs. *Rockyou* and *Phpbb* datasets are excluded in the following analysis, as we don't have access to user IDs/emails.

We identify *Counter* IDs utilizing Density-based Spatial Clustering of Applications with Noise (DBSCAN) [63]. DBSCAN [63] is a density-based clustering algorithm. It groups together points that are closely packed together. DBSCAN requires two parameters: ϵ and the minimum number of points required to form a dense region $minPts$. It starts with an arbitrary starting point that has not been visited. This point's $\epsilon-$neighborhood is retrieved, and if it contains sufficiently many points, a cluster is started. Otherwise, the point is labeled as noise. Note that this point might later be found in a sufficiently sized $\epsilon-$environment of a different point and hence be made part of a cluster. If a point is found to be a dense part of a cluster, its $\epsilon-$neighborhood is also part of that cluster. Hence, all points that are found within the $\epsilon-$neighborhood are added, as is their own $\epsilon-$neighborhood when they are also dense. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.

In our case, each ID is viewed as a point, and the distance between two IDs are measured by the Levenshtein distance, which measures the minimum number of single-character edits. Given a password, we first extract all the corresponding IDs in the dataset, and then generate groups of IDs, where the IDs in the same group share a common prefix with length at least 3. The grouping is introduced to reduce the number of points to be clustered, as calculating pairwise distance of a large number of data points is slow. Next, we apply

DBSCAN with $\epsilon = 1$ and $minPts = 5$ to each group. Finally, we label all IDs in clusters with size at least 5 as suspicious.

Random IDs are identified based on probabilities of IDs, which are calculated utilizing a Markov Model. Intuitively, *Random* IDs are ids whose probabilities are "unreasonably small". Observing that *Random* IDs are generally with the same length, for a fixed password, we propose to analyze the corresponding IDs with different lengths separately.



(a) English datasets (b) Chinese datasets

Figure 4.2.: CDF of $-\log_{10} p$ for all IDs with length 10 in all datasets. p is the probability of IDs. The dashed lines are CDF of normal distribution with the same mean and standard deviation

Figure 4.2 shows CDF of $-\log_{10} p$ of all IDs with length 10, which is the most frequently chosen ID length, from the 5 datasets containing IDs. For each dataset, we calculate probabilities of IDs utilizing 5-order Markov Model trained on itself. The dashed lines in the graph illustrate CDF of normal distribution with the same mean and standard deviation as the corresponding distribution. The figures show that except *178* dataset, the distributions of $-\log_{10} p$ fits relatively well with the corresponding normal distributions, especially for English datasets. The difference between the distribution from Chinese datasets, *178* in particular, and the corresponding normal distribution is larger, we believe the probability distribution of IDs in the dataset is biased by the problematic IDs. The graph indicates that the probabilities of IDs can be approximated by lognormal distribution.

Therefore, we perform "fake" account removal for IDs with the same length based on $-\log p$, where p is probabilities of IDs. Note that in a normal distribution, nearly all values are within three standard deviations of the mean (three-sigma rule of thumb), we therefore, believe $\mu + 3\sigma$ is a reasonable upper-bound for "real" IDs, where μ and σ are mean and standard deviation of $-\log p$, respectively.

In addition, if most of the IDs corresponding to a high-frequency password P in dataset D are detected as suspicious, and P does not appear in password datasets other than D, we remove all accounts associated with the P.

Table 4.6.: Top 10 passwords with most accounts removed from English datasets. $pwd_{r/o}$ means the original count of pwd in the dataset is o, and r accounts are removed.

Rank	*Yahoo*	*Xato*
1	1a1a1a1b$_{131/131}$	klaster$_{1705/1705}$
2	welcome$_{101/437}$	iwantu$_{885/885}$
3	-	1232323q$_{450/450}$
4	-	galore$_{393/393}$
5	-	wrinkle1$_{243/243}$
6	-	sex4me$_{229/229}$
7	-	Mailcreated5240$_{183/183}$
8	-	butler$_{182/182}$
9	-	meridian$_{180/180}$
10	-	finish$_{178/178}$

Table 4.7.: Top 10 passwords with most accounts removed from Chinese datasets. $pwd_{r/o}$ means the original count of pwd in the dataset is o, and r accounts are removed.

Rank	*Duduniu*	*Csdn*	178
1	aaaaaa$_{3103/10838}$	dearbook$_{44636/44636}$	qiulaobai$_{57963/57963}$
2	111111$_{1203/21763}$	xiazhili$_{3649/3649}$	wmsxie123$_{48258/49162}$
3	123456$_{1076/93259}$	12345678$_{2222/212743}$	123456$_{47536/261692}$
4	9958123$_{461/3981}$	123456789$_{1482/234997}$	w2w2w2$_{35762/35762}$
5	a5633168$_{457/457}$	11111111$_{1301/76340}$	wolf8637$_{31909/31909}$
6	woshi912$_{416/416}$	code8925$_{1285/1285}$	5508386$_{25715/25715}$
7	5ggggg$_{328/328}$	ms0083jxj$_{1268/1268}$	wpc000821$_{22733/22733}$
8	liuchang$_{319/2140}$	05962514787$_{910/910}$	111111$_{20564/122512}$
9	jkljkljkl0$_{281/346}$	google250$_{559/559}$	5897037$_{19266/19266}$
10	34537058gu$_{275/275}$	lilylily$_{395/765}$	js77777$_{18835/18835}$

Results of Cleansing. Table 4.5 lists the number of suspicious accounts removed. In general, the suspicious accounts count for a small portion in English and *Duduniu* datasets.

However, the number of suspicious accounts detected in *CSDN* and *178* datasets are significantly larger. In *178* dataset, about one fifth accounts are suspicious. Table 4.6 and Table 4.7 lists the top 10 passwords with most accounts removed in each dataset. Despite the accounts correspond to uncommon passwords, a significant number of accounts with popular passwords, such as `123456`, are removed as well. Evidences suggest that some datasets contain many waves of creation of suspicious accounts, some using common passwords such as `123456`, as illustrated in Table 4.4.

4.4 Experimental Results

4.4.1 Experimental Datasets and Settings

We evaluate PRAs on seven real-world password datasets, including four datasets from English users, *Rockyou* [50], *Phpbb* [50], *Yahoo* [50], and *Xato* [64], and three datasets from Chinese users, CSDN [60], *Duduniu*, and *178*.

Some PRAs require a training dataset for preprocessing. For English passwords, we train on *Rockyou* and evaluate on (1) *Yahoo* + *Phpbb*; (2) *Xato*, as *Rockyou* is the largest password dataset available. We combine *Yahoo* and *Phpbb* datasets because the size of them are relatively small. For Chinese passwords, the evaluation was conducted on any pair of datasets. For each pair, we trained PRAs on one dataset and tested on the other. We present results of using *Duduniu* as the training dataset.

Probabilistic Password Models. For all probabilistic password models we evaluate, we generate 10^8 passwords following the descending order of probabilities. The order of the password generated is essentially the ranking of the password in the corresponding PRA.

Blacklist PRAs. We directly use the training dataset as blacklist. Namely, in the PRA, the ranking of a password is the order of its frequency in the training dataset. We vary the size of blacklist by removing the lowest-rank passwords in order to adjust the number of passwords rejected.

Zxcvbn Password Generation. *Zxcvbn* was designed to evaluate password strength only. We implemented a password generator following the logic, which takes an input as the maximum entropy, and generates all passwords whose entropy is less than that value. The password generation is a recursive depth-first search process. We start from an empty string. In each iteration, we append a chunk to the current string. If the current entropy is less than the maximum entropy allowed, we record the password, and continue the recursion process. Note that we might duplicated generate passwords as each password might have multiple ways to be decomposed into patterns. Therefore, after the password generation, we conduct a further post-processing step. If a password appears multiple times, we keep the one with the lowest entropy, and then sort all the unique passwords based on entropy.

In practice, an integer score from 0 to 4 is calculated from entropy and each value is assigned with a description (e.g., Weak, Medium, Strong). Passwords with entropy less than 20 are assigned with a score is 0, and are usually rejected. We first tried to create all passwords within 20-bits of entropy. However, after 1 billion attempts, we still haven't finished generating passwords starting with "mary", which is the first word in female names dictionary. The number of passwords considered as weak by *zxcvbn* is much larger than any of the known weak password dictionary. Alternatively, we generated 10,478,853 unique passwords with entropy less than 4 for *zxcvbn*$_1$, and 1,834,980 unique passwords with entropy less than 12 for *zxcvbn*$_2$.

Adapting Zxcvbn to Chinese datasets. *Zxcvbn* was originally designed for English speaking users, as it supports English words only [23]. In order to adapt the method for evaluating Chinese passwords, we create another dictionary for evaluating Chinese passwords. We construct such a Chinese dictionary using the *Duduniu* dataset. For each password in *Duduniu*, we first split the word into chunks based on character types, e.g. letters, digits, and symbols. We then, count the frequencies of letter chunks after turning all letters into lower cases. Finally, we generate the dictionary by outputing all the letter chunks that contains at least three characters and with frequencies of at least 100 in the descending order of their frequencies. There are 5,553 words in the dictionary.

Table 4.8.: Top 20 words in the new dictionary for Chinese passwords used in *zxcvbn*.

Rank	Words				
1–5	asd	woaini	wang	abc	zhang
6–10	liu	qwe	love	qaz	yang
11–15	chen	zxc	aaa	wei	www
15–20	long	lin	xiao	aaaaaa	huang

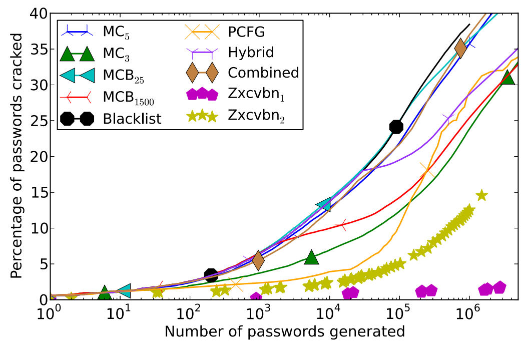
Table 4.8 lists the first 20 words in the reverse order of their frequencies. Most of the common words used are the syllables of last names in Chinese, e.g. wang, zhang, liu, etc. The rest of them are either keyboard patterns or letter sequences. Not many English words appears in the dictionary. There are many three letter combinations in the dictionary, such as wjq, ljh, zjh. We believe these are initials of the syllables in Chinese names. There is no need to construct separate name dictionaries as the most common names are already covered.

We were able to generate 9,316,973 passwords with entropy less than 3 for $zxcvbn_1$, and 1,913,061 unique passwords with entropy less than 12 for $zxcvbn_2$.

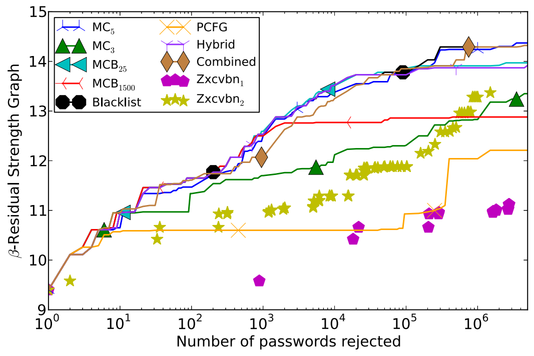
4.4.2 Experimental Results

Figure 4.3 and Figure 4.4 illustrates the Guess Number Graph (GNG), the β-Residual Strength Graph (β-RSG), and the Normalized β-Residual Strength Graph (β-NRSG) evaluated on *Xato* and *178* datasets, respectively. The corresponding training datasets are *Rockyou* and *Duduniu*. The evaluation on the other datasets leads to similar results.

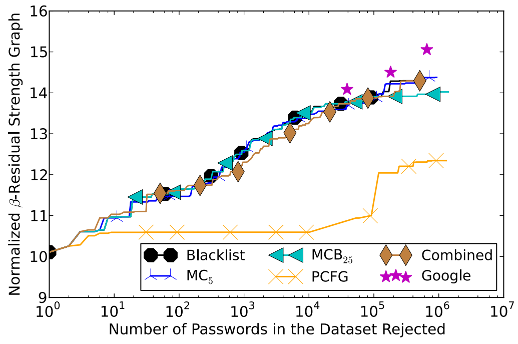
Figure 4.3(a) and Figure 4.4(a) show the evaluation of the Guess Number Graph (GNG). Both Client-end and Server-end PRAs, except Google's password strength assessment from which we are not able to generate passwords, are measured. We do not plot the Blacklist PRA with limited size, as it overlaps with the regular Blacklist PRA. We plot scatter points for *zxcvbn* to avoid ambiguity, since it generates multiple passwords with the same entropy.



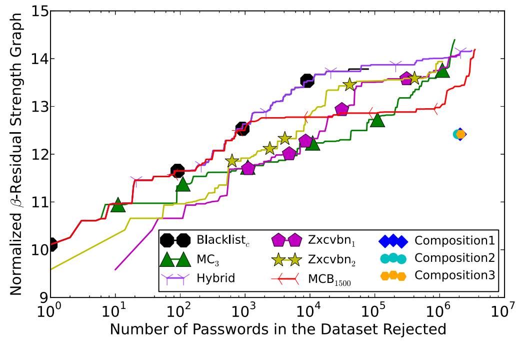
(a) GNG, *Xato*



(b) β-RSG, *Xato*, $\beta = 10$



(c) β-NRSG, *Xato*, Server-end, $\beta = 10$

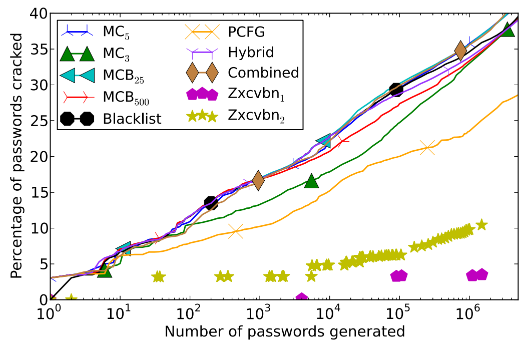


(d) β-NRSG, *Xato*, Client-end, $\beta = 10$

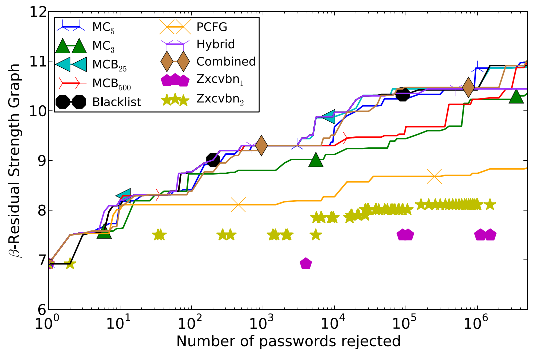
Figure 4.3.: The Guess Number Graph (GNG), the β-Residual Strength Graph (β-RSG), and the Normalized β-Residual Strength Graph (β-NRSG) evaluated on *Xato* dataset.

A point (x, y) on a curve means that y percent of passwords in the test dataset are included in the first x guesses.

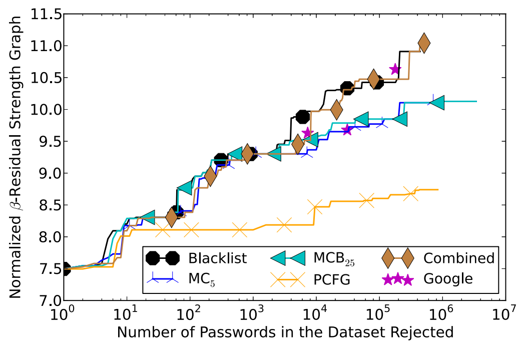
Figure 4.3(b) and Figure 4.4(b) illustrate the β-Residual Strength Graph (β-RSG) for $\beta = 10$. In the evaluation, we vary the number of passwords rejected x in PRAs (i.e., passwords ranked as top x are not allowed). In the figures, a point (x, y) on a curve means if we want to reject top x passwords from a PRA, the residual strength is y. For a fixed x, a larger y indicates smaller portion of accounts will be compromised within β guesses after rejecting x passwords. Comparing Figure 4.1(b) and Figure 4.4(b), we can observe that the performance of PRAs on cleansed data siginificantly boost, which confirm the need of data cleansing.



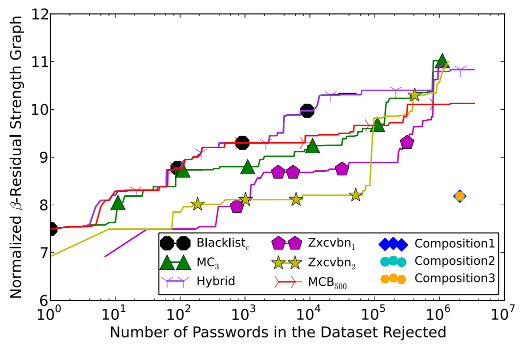
(a) GNG, *178*



(b) β-RSG, *178*, $\beta = 10$



(c) β-NRSG, *178*, Server-end, $\beta = 10$



(d) β-NRSG, *178*, Client-end, $\beta = 10$

Figure 4.4.: The Guess Number Graph (GNG), the β-Residual Strength Graph (β-RSG), and the Normalized β-Residual Strength Graph (β-NRSG) evaluated on *178* dataset.

The Normalized β-Residual Strength Graphs (β-NRSG) for Server-end PRAs are illustrated in Figure 4.3(c) and Figure 4.4(c), and the Client-end PRAs' evaluation is shown in Figure 4.3(d) and Figure 4.4(d). In addition to PRAs compared in GNG and β-RSG, we evaluate the effect of composition policies and Google's password strength API as well. Three commonly used composition rules are examined. Composition rule 1 is adapted by Ebay.com, which ask for at least two types of characters from digits, symbols and letter. Composition rule 2 is adapted by Live.com, which also ask for two types of characters, but it further split letters into uppercase and lowercase letters. Composition rule 3 is adapted by most of the online banking sites (e.g. BOA). At least one digit and one letter are required.

Server-end PRAs. In general, Server-end PRAs (Blacklist, MC_5, MCB_{25}, Combined) outperform Client-end PRAs (Hybrid, MC_3, MCB_{1500}/MCB_{500}), which confirms that a

PRA's accuracy grows with the size of its model, and Server-end PRAs are recommended for websites where security is one of the most important factors, e.g., online banks.

The Google password strength API, which is only evaluated in β-NRSG (Figure 4.3(c) and Figure 4.4(c)) is the top performer on both English and Chinese datasets. The three points from left to right in each graph illustrate the effect of forbidding passwords whose score is no larger than 1, 2, and 3, respectively. In practice, all passwords with score 1 are forbidden. The high residual strength indicates that most of the high-frequency passwords are successfully identified.

For the other Server-end PRAs, the three metrics (Figure 4.3(a, b, c) and Figure 4.4(a, b, c)) all suggest that several PRAs including the Blacklist PRA, the Markov Model with backoff with a frequency threshold of 25 (MCB_{25}), the 5-order Markov Model, and the Combined method [22] have similar performance, and they are almost always on the top of the graphs, which is consistent with the results in the previous works [22, 41].

Client-end PRAs. From Figure 4.3(d) and Figure 4.4(d), it is clear that composition rules do not help prevent weak passwords, as the corresponding points are far below the other curves. In addition, the composition rules generally reject more than one tenth of passwords in the datasets, which might lead to difficulty and confusion in password generation, and is not appropriate.

Among the other Client-end PRAs, the Blacklist PRA outperform the others when the number of passwords rejected is small. However, because of the limited size, the small blacklist can only cover a small proportion of passwords (less than 10,000) in the testing dataset. The reduced-size Markov models (MC_3 and MCB_c) perform siginificantly worse than the corresponding Server-end models (MC_5 and MCB_{25}), especially when the number of passwords rejected is relatively large. The low order Markov models cannot capture most of the features in the real passwords distribution and the strength measurement is not accurate. MCB_c performs similar to the Blacklist PRA when x is small, as the frequencies of the most popular patterns are high enough to be preserved, with the cost of losing most of the other precise information. As a result, the performance of MC_3 is better than MCB_c with the growth of x.

A noticable improvement of *zxcvbn*$_2$ over *zxcvbn*$_1$ can be observed in all the three metrics (Figure 4.3(a, b, d) and Figure 4.4(a, b, d)). The figures also suggest that *zxcvbn* is not optimized for passwords created by non-English speaking users, as the performance of the PRAs significantly drops in the evaluation on Chinese datasets.

The Hybrid Method. Observing the promising performance of Blacklist methods and the limited number passwords covered in the testing dataset, we propose a hybrid PRA which combines a blacklist PRA with a backoff model. In the Hybrid PRA, we first reject passwords based on the order in the Blacklist, and apply the backoff model after the Blacklist is exhausted. To make the size of the PRA consistent, we further limit the size for both the Blacklist and Markov Model with backoff. We set the frequency threshold to 2000 for the English password datasets and 1000 for the Chinese password datasets (see Table 4.2 for model sizes). We further reduce the size of the Blacklist to 30,000 words, resulting in a dictionary smaller than 300KB. The total size of the hybrid model is less than 800KB. The figures (Figure 4.3(a, b, d) and Figure 4.4(a, b, d)) show that the hybrid method inherits the advantage of Blacklist PRA and Markov Model with backoff. Hybrid method can accurately reject weak passwords, and can provide a relatively accurate strength assessment for any passwords. As a result, it is almost always on the top of all client-end PRAs, and is even comparable with Server-end PRAs in β-RSG and β-NRSG measurements.

Differences Among the Three Metrics. Table 4.9 and Table 4.10 list the y values in GNG and β-RSG when $x = 10^4$ and $x = 10^6$ evaluated on English datasets and Chinese datasets, respectively. From the tables, we can observe that although the percentage of passwords cracked by PRAs significantly increase from when rejecting ten thousand passwords to when rejecting one million passwords, the difference between y values in β-RSG is limited, especially for the top-performing PRAs, such as the blacklist method. The different behavior between GNG and β-RSG indicates that the percentage of passwords cracked, which is shown in GNG, cannot infer the residual strength, which is the observation from β-RSG. A high coverage and a low coverage in password cracking might result in similar residual strength, as the most frequent remaining passwords might be similar. The result from the table confirms that if the thread model is online guessing attacks in which the

Table 4.9.: y values of GNG and β-RSG when $x = 10^4$ and $x = 10^6$ evaluated on English datasets. *Y+P* stands for *Yahoo + Phpbb*. $\beta = 10$

	English Datasets							
	GNG				RSG			
Dataset	*Y+P*		*Xato*		*Y+P*		*Xato*	
x	10K	1M	10K	1M	10K	1M	10K	1M
MC_5	14%	34%	13%	36%	12.7	13.1	13.4	14.2
MC_3	7.3%	21%	6.9%	24%	11.5	12.4	12.1	12.8
MCB_{25}	16%	35%	14%	36%	12.8	13.2	13.5	13.9
MCB_c	11%	22%	10.0%	25%	12.6	12.7	12.8	12.9
zxcvbn	0.7%	1.4%	0.5%	1.3%	10.1	10.8	10.0	11.0
$zxcvbn_{v2}$	2.5%	13%	2.4%	13%	11.2	12.8	11.3	13.3
Blacklist	16%	38%	14%	38%	12.8	13.3	13.5	14.3
PCFG	2.2%	22%	3.9%	29%	10.2	11.9	10.6	12.0
Hybrid	16%	27%	14%	29%	12.8	13.0	13.5	13.9
Combined	13%	36%	13%	37%	12.8	13.2	13.2	14.3

Table 4.10.: y values of GNG and β-RSG when $x = 10^4$ and $x = 10^6$ evaluated on Chinese datasets. $\beta = 10$

	Chinese Datasets							
	GNG				RSG			
Dataset	*CSDN*		*178*		*CSDN*		*178*	
x	10K	1M	10K	1M	10K	1M	10K	1M
MC_5	16%	26%	22%	36%	10.2	10.3	9.7	10.9
MC_3	13%	23%	18%	33%	9.7	9.9	9.1	10.2
MCB_{25}	17%	27%	23%	36%	10.3	10.4	9.9	10.4
MCB_c	16%	25%	21%	33%	10.1	10.3	9.3	10.2
zxcvbn	0.1%	3.5%	0.3%	3.4%	6.6	7.1	7.0	7.5
$zxcvbn_{v2}$	3.5%	11%	4.8%	9.8%	7.1	8.9	7.8	8.1
Blacklist	17%	26%	23%	35%	10.3	10.3	10.0	10.4
PCFG	16%	21%	15%	24%	9.7	9.8	8.3	8.7
Hybrid	17%	25%	23%	34%	10.3	10.3	10.0	10.4
Combined	17%	27%	22%	36%	10.3	10.3	9.5	10.5

number of attempts allowed by an adversary is limited, GNG cannot accurately measure the crack-resistency of PRAs, and β-RSG is a more appropriate metrics in this use case. The low marginal effect in β-RSG also indicates that websites might not need to reject too many passwords if the major concern is online guessing attacks.

From Figure 4.3 and Figure 4.4, perhaps the most noticable difference among the metrics is the relative order of the PCFG method, two versions of *zxcvbn*, and the Hybrid method, compared with the other Client-end PRAs.

The PCFG method performs reasonably well in GNG, but poorly in β-RSG and β-NRSG. While PCFG can cover many passwords in the testing datasets, which leads to the low total density of passwords not cracked in GNG, some of the high-frequency passwords remain uncovered. As a result, the residual strength of PCFG is lower than most of the other PRAs.

On the other hand, the hybrid method and *zxcvbn*$_2$ perform much better in β-RSG and β-NRSG than in GNG. Although the high-ranking passwords in the PRAs only include a relative low number of unique passwords in the testing datasets, the popularly selected passwords are mostly covered. Therefore, after rejecting the top-ranking passwords from the PRAs, an adversary can only break into a limited number of accounts within a small number of guesses, which results in a high residual strength.

Another obervation is that the performance of the two *zxcvbn* PRAs, especially *zxcvbn*$_2$ significantly boost in β-NRSG comparing with that in β-RSG. The residual strength resulted by *zxcvbn*$_2$ is even higher than the size-limited Markov Models (MC_3 and MCB_c). The observation indicates that the relative poor performance of *zxcvbn* in β-RSG is mainly due to the penalization from the large number of passwords, which are extremely not likely to be used, generated.

Overall, several Server-end PRAs including the Blacklist PRA, the Markov Models, and the Combined method result in similar perfomances. The hybrid method, which inherits the advantage of Blacklist PRAs and Markov Model with backoff, outperform the other Client-end PRAs.

4.5 Conclusion

In this chapter, we model different password strength checking methods (including password strength meters) as Password Ranking Algorithms (PRAs), and we introduce two metrics: the β-Residual Strength Graph (β-RSG) and the Normalized β-Residual Strength Graph (β-NRSG), to compare them using real world password datasets. In our evaluation, we find unreasonably high frequency of some suspicious passwords. We remove the associated accounts by identifying suspicious account IDs. We then, apply the metrics on cleansed datasets, and show that dictionary-based PRA has similar performance with the sophisticated PRAs. If the size of PRAs are limited in order to be fit into a client, a hybrid method combining a small dictionary of weak passwords and a Markov Model with backoff with a limited size can provide the most accurate strength measurement.

5. UNDERSTANDING MNEMONIC SENTENCE-BASED PASSWORD GENERATION STRATEGIES

Besides weak passwords checking by websites/servers, another way to avoid weak passwords is to educate users in password creation techniques, i.e, password generation strategies, and help users creating secure and memorable passwords.

Perhaps the most widely recommended and studied strategy is that based on mnemonic sentences: Take a memorable sentence, abbreviate the words, and combine them to form a password. The strategy is generally known as the mnemonic sentence-based strategy (for short, the *mnemonic strategy*).

In this chapter, We evaluated the security of 6 mnemonic strategy variants in a series of online studies involving 5484 participants. In addition to applying the standard method of using guess numbers or similar metrics to compare the resulted passwords, we also measured the total density of the most frequently chosen sentences as well as the resulting passwords. While metrics similar to guess numbers suggest that all variants provided highly secure passwords, statistical metrics told a more interesting and nuanced story. In particular, differences in the exact instructions had a tremendous impact on the security level of the resulted passwords. We examined the generation usability and memorability of mnemonic strategies in another online study with 1265 participants. We found that differences in the exact instructions also affect the usability of the mnemonic strategies, and a tradeoff between security and memorability was observed.

5.1 Introduction

The security community has been trying to come up with password generation strategies that can help users generate secure and usable passwords. Candidate strategies have been suggested by sources ranging from the National Institute of Standards and Technology

(NIST) [24] to online comics [25], and from security experts' essays [26, 27] to online help forums. However, these suggestions are often based on intuitions instead of scientific knowledge. Little is actually known about which strategies are effective in helping users create usable and secure passwords.

Perhaps the most widely recommended and studied strategy is that based on mnemonic sentences: Take a memorable sentence, abbreviate the words, and combine them to form a password. The strategy is generally known as the mnemonic sentence-based strategy (for short, the *mnemonic strategy*). It appears that the general assessment is that this is a good strategy. It is recommended by NIST [24] and by security experts [26, 27]. To our knowledge, three studies on this method have been reported, by Yan et al. [46, 47], Vu et al. [48], and Kuo et al. [49]. One standard approach for evaluating the strength of passwords is to use password cracking tools or models to check how many collected passwords can be cracked [8, 9, 11, 42, 45, 65]. Based on this, Yan et al. [46, 47] claimed that passwords generated using the mnemonic strategy are as strong as random passwords, while others reached a somewhat mixed conclusion regarding its security [48, 49].

These existing studies, however, have several limitations. First, they are based on samples of small sizes, with less than 150 passwords under the strategy in each of the three studies. Second, relying only on checking how many passwords can be cracked to assess the security is flawed. Although such assessment provides useful information about how such passwords fare against today's state of the art cracking methods, the results are often caused by the incompatibility of the cracking techniques and the nature of the passwords, as in the case of evaluating the mnemonic strategies. Even developing a strategy-specific cracking method, as done in [49], is insufficient. It is always possible that one has overlooked some highly effective attack techniques. Third, each study considers only one version of instructions for the mnemonic strategy.

We conduct a much larger study to evaluate 6 variants of the mnemonic strategy and compare against a control group. When assessing the security of the strategies, we go beyond the methods used in existing studies in two ways. First, we adopt the approach of using statistical quantities to measure the distributions of the passwords, as articulated by

Bonneau [28]. In particular, we chose to use the *β-guess-rate* (λ_β) [29], which measures the expected success for an attacker limited to β guesses per account. As our dataset sizes (close to 800) are much smaller than the ones studied in [29], many other metrics considered in [29] cannot be applied. We chose to use $\beta = 1$ and $\beta = 10$, both because they were suggested in [30] as appropriate for defense against online guessing attacks and because a larger β is not very meaningful for our sample sizes. Second, we develop a method specific for attacking passwords resulted from the mnemonic strategy, and demonstrate the effectiveness of this attack.

The usability of the variants is evaluated in a separate user study, in which password creation time, short-term and long-term password memorability, and the workload required in both password creation and retention are evaluated.

Our studies were conducted on Amazon Mechanical Turk. They were found to be eligible for exemption from IRB review because it is research involving survey procedures, and human subjects cannot be identified from the recorded information. In the studies, participants were warned not to use their real passwords.

To the best of our knowledge, we are the first to investigate the security of password generation strategy variants on a large scale. We recruited a total number of $5,484$ participants, for an average of 783 participants per condition, when evaluating the security of the strategies. In addition, we recruited 1265 participants for evaluating the usability of two variants against a control condition. To our knowledge, we are the first to observe and experimentally validate the influence of the instructions and the examples accompanying the strategy description on the security of the passwords generated with that strategy. It is intuitively understood that precise instructions and demonstrative examples can improve the ease of applying a strategy to generate passwords (i.e., usability). However, the relationship between the level of security, the instruction wording, and the examples has not been studied before.

5.2 Study 1: Security

In this section, we present an overview of the first study and the methodology used for evaluating security of the strategies.

We study 6 variants of mnemonic strategies. In such a strategy, a participant is asked to first select an easy-to-remember sentence, and then convert the sentence into a password.

Table 5.1.: Mnemonic-based strategy description

Strategy	Short Description	Exact instruction given to the users in the study
MneGenEx	Mnemonic with generic example, used in Kuo et al. [49]	1. Think of a memorable sentence or phrase containing at least seven or eight words. For example, "*Four score and seven years ago our fathers brought forth on this continent*". 2. Select a letter, number, or a special character to represent each word. A common method is to use the first letter of every word.For example: four –>4, score –>s, and –>&. Combine them into a password: 4s&7yaofb4otc.
MnePerEx	Mnemonic with emphasis on personalization, with an example.	1. Think of a memorable sentence or phrase that is meaningful to you, and other people are unlikely to use. The sentence or phrase should contain at least eight words. For example, "*I went to London four and a half years ago*". 2. Select a letter, number, or a special character to represent each word. A common method is to use the first letter of every word. For example: went ⇒ w, four ⇒ 4, and ⇒ &. Combine them into a password: iwtl4&ahya.

continued on next page

Table 5.1.: *continued*

Strategy	Short Description	Exact instruction given to the users in the study
MnePer	Mnemonic with emphasis on personalization, without giving an concrete example.	1. Think of a memorable sentence or phrase that is meaningful to you, and other people are unlikely to use. The sentence or phrase should contain at least eight words. 2. Select a letter, number, or a special character to represent each word, and combine them to create the password.

continued on next page

Table 5.1.: *continued*

Strategy	Short Description	Exact instruction given to the users in the study
MneEx	Mnemonic with several personalized phrases as examples.	1. Think of a memorable sentence or phrase containing at least eight words. 2. Select a letter, number, or a special character to represent each word, and combine them to create the password. The following are some examples: "*In June 2013, my wife and I visited Tokyo, Kyoto, and Sapporo*" might become "i63mw&ivTk&$". "*Run 5 miles per week for my first half marathon*" might become "r5mpw4mfhm". "*My high school classmates had a reunion in July 2014*" might become "Mhscharij2". "*I sold my gold Toyota corolla when it had close to 120000 miles*" might become "i$mgtcwIhc21m". "*Danny bought the book The Razor's Edge from me for five dollars*" might become "Dbtbtrefm45d". "*Save money for traveling with my parents to Germany*" might become "S$4twmp2G".

continued on next page

Table 5.1.: *continued*

Strategy	Short Description	Exact instruction given to the users in the study
MneSchEx	Mnemonic with mixed examples, used in [26]	1. First create a personally memorable sentence (choose your own sentence – something personal). 2. Then use some personally memorable tricks to modify that sentence into a password. The following are some examples: "*This little piggy went to market*" might become "tlpWENT2m". "*When I was seven, my sister threw my stuffed rabbit in the toilet*" might become "WIw7,mstmsritt". "*Wow, does that couch smell terrible*" might become "Wow...doestcst". "*Long time ago in a galaxy not far away at all*" might become "Ltime@go-inag faaa!". "*Until this very moment, these passwords were still secure*" might become "utvm,tpwstillsecure".

continued on next page

Table 5.1.: *continued*

Strategy	Short Description	Exact instruction given to the users in the study
MneYanEx	Mnemonic with mixed examples, used in [46, 47].	1. Please create a simple sentence of 8 words and choose letters from the words to make up a password. You should put some letters in upper case to make the password harder to guess; and at least one number and/or special character should be inserted as well. 2. Use this method to generate a password of 7 or 8 characters. An example of such a composition might be using the phrase is "*It's 12 noon I am hungry*" to create the password "I's12&Iah" which is hard for anyone else to guess but easy for you to remember. By all means use a foreign language if you know one: the password "AwKdk.Md" from the phrase "*Anata wa Kyuuketsuki desu ka ... Miyu desu*" would be an example. You could even mix words from several languages. However, do not just use a word or a name from a foreign language.

Table 5.1 gives the detailed descriptions of the 6 variants in our study. We urge readers to read Table 5.1 before proceeding, as the differences in the strategy descriptions are an important part of the study. Below is a summary.

- MneGenEx (Mnemonic-Generic-Example, with generic instruction and a generic example, similar to what used in Kuo et al. [49]),

- MnePerEx (Mnemonic-Personalized-Example, with emphasis on using personalized choices of sentences that other people are unlikely to use and a personalized example),
- MnePer (Mnemonic-Personalized, with emphasis on personalized choice of sentences, but no example),
- MneEx (Mnemonic-Example, with multiple personalized examples, but no emphasis on personalized choices of sentences),
- MneSchEx (Mnemonic-Schneier-Example, with some emphasis on personalized choices and mixed examples, suggested by Schneier in [26, 27]),
- MneYanEx (Mnemonic-Yan-Example, with some emphasis on personalized choices in some examples, used by Yan et al. [46, 47] in their studies).

In addition to the 6 variants, a control group Control, in which we ask for passwords containing at least 8 characters without any extra restriction, was included in the study as well.

5.2.1 Study Design

We conducted the study through Amazon Mechanical Turk (MTurk), and all participants were at least 18 years old. We limited our data collection to participants from the United State because the strategies were constructed using the English language. The study was divided into 7 rounds, one for each of the 7 conditions. Participants were allowed to participate in only one round. If a participant was in more than one rounds of the study, we kept only the data from the first time that the participant was in.

Participants were asked to type the sentence used in the intermediate step. After that, participants were asked to enter the password they created twice, and the password typed in each time had to match each other before they could proceed.

We warned participants not to use their actual passwords. In the study, we forbade passwords that were the same as the examples and that did not appear to be generated following the instructions. In MneGenEx, MnePerEx, MnePer, and MneEx, we required the length of the password to be identical to the number of words, and further checked if a letter in a password can be found in the corresponding word in the sentence; no check was performed for digits and special symbols in the password. In MneSchEx and MneYanEx, a participant was allowed to keep a complete word in the resulting password; thus we cannot use the above approach. Instead, we required that the sequence of letters (ignoring special symbols or digits) in a password was a subsequence of the sequence of letters in the sentence.

5.2.2 Methodology

Our goal in this study is to assess the security of the passwords generated by the different password generation strategy variants. The traditional approach to assess the strength of passwords generated under a given setting is to use password cracking tools or probabilistic password models to plot guess number graphs [8,9,11,42,45,49,65] or probability threshold graphs.

The above approach can assess the security of passwords against current password cracking tools and probabilistic password models, which are adapted to today's password distributions; however, it cannot adequately assess the strength of password strategies against attacks targeting it. If any strategy is widely used, then attackers may develop strategy-specific methods which can efficiently guess the passwords. For example, if the mnemonic strategy is widely used, one attack strategy is conduct a dictionary attack which use password datasets created using the strategy as the dictionary. Alternatively, an adversary can create a dictionary of sentences that people are likely to use, and then generate guesses from the sentences.

An alternative approach, as articulated by Bonneau [28], is to measure the probability distribution induced by the strategy. A number of metrics on the strength of password distributions have been proposed by Bonneau [28]. In the case of evaluating passwords obtained from user subject studies, the datasets are quite small (on the scale of several hundreds in our case). One metric that is appropriate for small datasets is the *β-guess-rate* (λ_β) [29], which is the total probability of the most common β passwords. λ_β measures the expected success for an attacker limited to β guesses per account, with some small β. Brostoff and Sasse [30] suggested 10 as the allowed failure counts before the account is locked.

We choose to use quantities for estimating λ_1 and λ_{10} given our sample sizes. Given a sample set S, we use $\mathsf{top}(S)$ to denote the number of times that the most frequent item occurs in S, and $\mathsf{top}_{10}(S)$ to denote the total number of the times the 10 most frequent items occur in S. The estimated density of top 1 and top 10 passwords are calculated by $\tilde{\lambda}_1 = \frac{\mathsf{top(S)}}{|S|}$ and $\tilde{\lambda}_{10} = \frac{\mathsf{top_{10}(S)}}{|S|}$, where $|S|$ is the size of S.

We want to tell whether the differences in these metrics between two datasets are meaningful or not, given that we have small datasets. To address the issue, for a given β, we test the null hypothesis that the total density of top β passwords in the two datasets are the same using the *two proportion z-test*, calculated by:

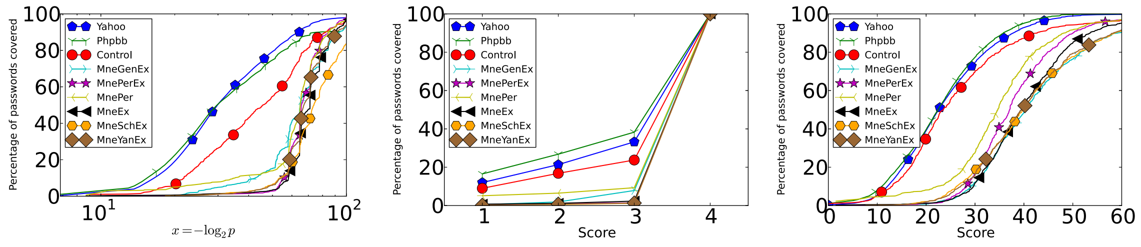
$$z = \frac{p_1 - p_2}{\sqrt{p(1-p)(\frac{1}{n_1} + \frac{1}{n_2})}}$$

where $p_1 = \frac{x_1}{n_1}$ and $p_2 = \frac{x_2}{n_2}$ are the two proportions from the two samples, i.e., total density of top β passwords in two datasets, D_1 and D_2; n_1 and n_2 are the size of D_1 and D_2; p is pooled sample proportion, which is estimated by $\frac{x_1+x_2}{n_1+n_2}$; and x_1 and x_2 are the total frequencies of top β passwords in D_1 and D_2.

We also apply these metrics to the sentences used in generating passwords, because passwords based on the same sentence are not independent of each other.

5.3 Results From Study 1

We recruited 864, 793, 797, 795, 982, and 870 participants for the 6 variants of mnemonic strategies. After removing duplicate participants, the number of participants we accepted was 864, 777, 753, 745, 868, and 799, respectively. The number of participants recruited in the control group (Control) is 678. In total, 5,484 (3,205 female) participants were involved. The participants' ages ranged from 18 to over 50, with about 70% between 23 to 50 years. Most of the participants were college students or professionals who had bachelor or higher degrees. The demographic distributions in the 7 groups were similar.



(a) 5-order Markov Model (b) Google API (c) Zxcvbn

Figure 5.1.: Comparison of strength of passwords resulted from different mnemonic strategies using probabilistic models and password strength meters.

5.3.1 Analyzing Passwords Using Probability Models and Password Strength Meters.

We evaluate the strength of passwords generated using the strategies as well as two commonly used datasets Yahoo and Phpbb against today's attacks utilizing (1) the 5-order Markov Model trained on Rockyou dataset, (2) Google password strength API[1], and (3) Zxcvbn [23] deployed by Dropbox. Google password strength API produces an integer score from 1 to 4 for a password. Passwords with score 1 are considered too weak and are forbidden by Google, and passwords with score 4 are considered strong. Zxcvbn gives an estimation of minimum entropy for a password. The entropy is calculated by first dividing the password into chunks, and then combining the entropy estimated for each chunk. Different ways of dividing the password results in different estimated entropy, and Zxcvbn uses the smallest entropy as its output.

In Fig 5.1(a), each curve conveys the strength of a password dataset evaluated by a 5-order Markov Model trained on Rockyou dataset. A point (x, y) on a curve means that

[1]https://accounts.google.com/RatePassword

in the corresponding dataset, y percentage of passwords have probability no less than 2^{-x}. Curves in Fig 5.1(b) and Fig 5.1(c) illustrate the evaluation based on Google password strength API and zxcvbn, respectively. A point (x, y) on a curve means y percentage passwords in the corresponding dataset has a score no higher than x. In the graphs, a lower curve means passwords from the corresponding strategy are considered stronger.

In all the three graphs, the curve for the control group (Control) is below the curves for Yahoo and Phpbb, indicating that passwords created in the study are stronger than that in real-world datasets. Therefore, the security of passwords created in the study can serve as a lower-bound measurement. On the other hand, curves for Yahoo, Phpbb, and Control are significantly higher than the other curves. This indicates that according to the metrics, passwords created without any specific strategy are significantly weaker than the other datasets. When guessing according to the order suggested by the metrics, more passwords in Yahoo, phpbb, and Control will be cracked than passwords from any mnemonic strategy. For example, if all passwords with score less than 25 measured by Zxcvbn are attempted, more than 50% of passwords from Yahoo, Phpbb, and Control will be covered, while in the 6 mnemonic strategies, the percentage of passwords cracked is less than 15%. However, this conclusion is due to the fact that the model or the meters are designed to evaluate generally selected passwords, which is broadly similar to passwords in Yahoo and Phpbb, and passwords generated from the mnemonic strategies result in quite different distributions.

Table 5.2.: Average length of passwords in each strategy as well as Yahoo and Phpbb datasets.

Strategy/Dataset	Average Length
Yahoo	7.6
Phpbb	8.3
Control	10.4
MneGenEx	10.1
MnePerEx	9.2
MnePer	9.1
MneEx	9.4
MneSchEx	11.4
MneYanEx	9.6

Table 5.2 shows the average length of passwords generated from the 6 variants and the control group (Control) as well as passwords in Yahoo and Phpbb datasets. From the table, we can observe that passwords generated in MneSchEx is longer than passwords generated by other strategies. Another observation is the relative long passwords in Control comparing with passwords in Yahoo and Phpbb datasets. The average length of passwords in Control is even longer than passwords from some variants of mnemonic-based strategies, which confirms that passwords created in the study is stronger than typical users' choices.

Table 5.3.: $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) in Control as well as samples with size 800 from Rockyou, Phpbb, and Yahoo. E_{SD} means the average is E and the standard deviation is SD.

Strategy	Count	$\tilde{\lambda}_1$ (top)		$\tilde{\lambda}_{10}$ (top$_{10}$)	
		Case Insensitive	Case Sensitive	Case Insensitive	Case Sensitive
Control	678	1.2%(8)	0.9%(6)	3.4%(23)	2.9%(20)
Rockyou^s	800	$1.0\%_{0.3\%}(7.7)$	$0.9\%_{0.4\%}(7.5)$	$3.1\%_{0.5\%}(25.0)$	$3.1\%_{0.5\%}(24.4)$
Phpbb^s	800	$1.2\%_{0.4\%}(9.5)$	$1.2\%_{0.4\%}(9.5)$	$3.8\%_{0.6\%}(30.2)$	$3.8\%_{0.5\%}(30.2)$
Yahoo^s	800	$0.4\%_{0.2\%}(3.5)$	$0.4\%_{0.2\%}(3.5)$	$2.1\%_{0.3\%}(16.5)$	$2.0\%_{0.3\%}(16.3)$

5.3.2 Strength of Mnemonic Sentences

We evaluate the strength of sentences used in mnemonic strategies as well as the resulted passwords utilizing $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ metrics. Table 5.3 shows the $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) values of passwords generated by the control group (Control). Also shown in the table are the quantities evaluated on sets of 800 passwords randomly sampled from three commonly used password datasets Rockyou, Phpbb and Yahoo. Table 5.4 gives the $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) values of sentences and the resulted passwords for all mnemonic sentence-based variants.

The Control Group (Control). In Control, the $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ are 0.9% and 2.9%, respectively, which are close to the quantities from the real-world datasets. For instance, Rockyou dataset has a $\tilde{\lambda}_1$ as 0.9% and a $\tilde{\lambda}_{10}$ as 3.1%. Although the passwords created in the study

Table 5.4.: $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) in mnemonic strategies.

Strategy	Count	$\tilde{\lambda}_1$ (top)			$\tilde{\lambda}_{10}$ (top$_{10}$)		
		Sentence	Password		Sentence	Password	
			Case Insensitive	Case Sensitive		Case Insensitive	Case Sensitive
MneGenEx	864	2.5%(22)	0.9%(8)	0.8%(7)	7.8%(68)	5.3%(46)	4.1%(36)
MnePerEx	777	0.1%(1)	0.1%(1)	0.1%(1)	1.3%(10)	1.3%(10)	1.3%(10)
MnePer	745	0.7%(5)	2.3%(17)	2.3%(17)	2.8%(21)	5.8%(43)	5.6%(42)
MneEx	868	0.7%(6)	0.2%(2)	0.2%(2)	2.2%(19)	1.7%(15)	1.3%(11)
MneSchEx	753	0.4%(3)	0.5%(4)	0.3%(2)	2.8%(21)	1.7%(13)	1.5%(11)
MneYanEx	799	0.3%(2)	0.3%(2)	0.3%(2)	1.6%(13)	1.5%(12)	1.4%(11)
Control	678	N/A	1.2%(8)	0.9%(6)	N/A	3.4%(23)	2.9%(20)

are stronger than those in real-world datasets, according to the existing strength metrics, as illustrated in Fig 5.1, the strengths of the weakest passwords are similar. If an adversary is limited to try 10 passwords per account (e.g., by rate limiting), approximately the same portion of accounts will be compromised.

Finding 1: Using generic instructions and examples results in weak passwords. MneGenEx used the instructions as in [49] and one of the examples used in [49]. We were truely surprised by the high frequencies of the most common sentences and passwords. Among the 864 participants, there were 57 sentences chosen more than once, for a total of 179, and the 10 most popular sentences (top_{10}) were picked 68 times. 22 participants chose the famous quote "*to be or not to be, that is the question*". This yielded $\tilde{\lambda}_1 = 2.5\%$, $\tilde{\lambda}_{10} = 7.8\%$. See Table 5.5 for other commonly chosen sentences, which were also well-known quotes in general, and the resulted passwords.

Table 5.5.: Popular passwords and probability for top 5 frequently chosen sentences in mnemonic strategies.

Rank	Sentences	Passwords	Frequency
MneGenEx (864)			
1	to be or not to be, that is the question (22)	2bon2btit? (7); 2bon2btitq (6); tbontbtitq (1); 2Bon2Btit? (1); 2B0n2bt1tq (1); 2bontbtitq (1); 2brn2btstq (1); 2brn2btit? (1)	2.55%
2	the quick brown fox jumped over the lazy dog (9)	tqbfjotld (2); Tqbfjotld (2); t@bfj0tld (1); tqb4j0tld (1); TQ35j#TLd (1); tqbfjîld (1); Tq8fj0tld (1)	1.04%
3	one small step for man, one giant leap for mankind (6)	1ssfm1glfm (3); 1ss4m1gl4m (1); 1$$4m1gl4m (1); ossfmoglfm (1)	0.69%

continued on next page

Table 5.5.: *continued*

Rank	Sentences	Passwords	Frequency
4	a penny saved is a penny earned (5)	apsiape (3); @p$i@p3 (1); apsi-Ape (1)	0.58%
5	in the beginning, god created the heavens and the earth (5)	itbGcth&te (1); 1t8GctH&t3 (1); it-bGcth&tE (1); ItbGctHatE (1); Nt-bGcth (1)	0.58%
MnePerEx (777) No collisions found.			
MnePer (745)			
1	I love you to the moon and back (4)	12345678 (1); !l0t7m@b (1); ily2tmnb (1); !@#$%^&* (1)	0.67%
2	it was the best of times it was the worst of times (3)	iwtb0*iwtw0* (1); Iwtbotiwt-wot (1); 233425233525 (1)	0.40%
3	the quick brown fox jumped over the lazy dog (2)	tqbfjotld (2)	0.27%
4	don't look a gifthorse in the mouth (2)	dlaghitm (1); d*1gh0t% (1);	0.27%
5	down by the bay where the watermelons grow (2)	dbhaw!rg (1); DBTBWTWG (1)	0.27%
MneEx (868)			
1	the quick brown fox jumped over the lazy dog (6)	tqbfjotld (1); 7qbxj07ld (1); tQbfj0tld (1); +qbfj0+ld (1); tqbfj0tld (1); tQbfôtzd (1)	0.69%
2	to be or not to be that is the question (3)	2Bon2Btit? (1); 2bontbtitq (1); 2bon2b,it? (1)	0.35%

continued on next page

Table 5.5.: *continued*

Rank	Sentences	Passwords	Frequency
3	my very educated mother just served us nine pizzas (2)	Mvemj$u9p (1); mvemjsu9p (1)	0.23%
4	I like big butts and I cannot lie (2)	1lbba1cl (1); 1lbb&1cnl (1);	0.23%
MneSchEx (753)			
1	four score and seven years ago (3)	4score7yo (1); foscanseyeag (1); fscrn7yrg (1)	0.40%
2	the quick brown fox jumps over the lazy dog (3)	tqbFOXjotlDOG (1); tqbfjotld (1); Tqbfjotld (1)	0.40%
3	once upon a time (2)	O345$&on@tim8 (1); 1ce-upontme (1)	0.27%
4	I love to eat pizza (2)	eyeL2EZa (1); ILtePi&&a (1);	0.27%
5	I love dark chocolate (2)	eyeluvdrkchoco (1); heartDlate (1);	0.27%
MneYanEx (799)			
1	the quick brown fox jumped over the lazy dog (2)	tqbfjotld (2);	0.25%
2	i like big butts and i cannot lie (2)	ilbbaicl (1); Ilbbaicl (1);	0.25%
3	the quick brown fox jumped over the dog (2)	Tqbf&jotD (1); TQbfdreg (1);	0.25%

In terms of resulting passwords, if passwords are case-insensitive, 36 passwords generated by following the MneGenEx strategy appeared more than once, and the most common password was chosen 8 times, with $\tilde{\lambda}_{10} = 5.3\%$. Even taking case-sensitivity into account, there were still 27 non-unique passwords, with the top count number to be 7, and $\tilde{\lambda}_{10} = 3.1\%$, as the majority of the participants did not use upper-case letters. Comparing $\tilde{\lambda}_{10}$ resulted from Control (2.9%) and MneGenEx (4.1%), it appears that the password distribution resulted from MneGenEx is likely to be weaker than Control.

Finding 2: Instructions specifically requesting personalized sentences and appropriate examples lead to strong passwords. MnePerEx explicitly asked people to choose personalized sentences that other people are unlikely to choose with an example "I went to London four and a half years ago". Among the 777 participants, there was no sentence or password selected more than once. We observed that 536 sentences start with "I" or "my", suggesting a personalized choice. In comparison, such sentences appeared only 125 times in MneGenEx. We noted that not all participants chose personalized sentences. Common sentences such as "to be or not to be, that is the question" still occur in the dataset. Because they occur with much lower frequencies, we did not observe any collision in the dataset. With larger datasets, collisions are bound to occur. As a result, the $\tilde{\lambda}_{10}$ (1.3%) in sentences selected in MnePerEx was significantly smaller than the quantity from MneGenEx ($z = 6.26, p < 0.001$), and the comparison of the resulted passwords between MneGenEx and MnePerEx leads to similar results. This indicates that in terms of security, MnePerEx is significantly better than MneGenEx based on $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ metrics.

Finding 3: Commonly suggested instantiations are worse than MnePerEx. Seeing results from MneGenEx and MnePerEx, it was clear to us that the instructions played an extremely critical role in the level of security. We then tried to evaluate the precise instructions suggested in Bruce Schneier's two blog posts [26, 27]. We noted that the instructions in the two posts were slightly different. Our version, MneSchEx, was based on the version in [26], which was the more elaborated one. MneSchEx had several differences from MnePerEx. First, it gave 4 examples, some of which are popular, e.g., "Long time ago in a galaxy not far away at all", others are more personalized "When I was seven, my sister threw my stuffed rabbit in the toilet". Second, in the examples, some words are completely kept. Third, while the instructions said "Choose your own sentence – something personal"; it did not include the phrase "other people are unlikely to use".

The results came back at somewhere in between MneGenEx and MnePerEx. Among 753 participants, 9 different sentences were not uniquely chosen, with the most common sentence appearing 3 times and the $\tilde{\lambda}_1$ was 0.4%. The $\tilde{\lambda}_{10}$ of sentence selected was 2.8%. There was only a single password selected twice. The $\tilde{\lambda}_{10}$ from MneSchEx was significantly larger than that from MneGenEx ($z = 4.47, p < 0.001$), and was significantly smaller than MnePerEx ($z = 2.08, p = 0.019$). One might notice that if passwords are not case-sensitive, the frequency of the most common password was more than the max count of sentences. The four repeated passwords actually came from 3 variations of the same sentence "*the quick brown fox jumps over the lazy dog*", "*the quick brown fox jumped over the lazy dogs*", and "*the quick brown fox jumped over the lazy dog*".

We also studied the effect of the instructions and examples used in Yan et al. [46, 47] (MneYanEx). In MneYanEx, the instructions for creating passwords was relatively generic. However, "hard for anyone else to guess" was explicitly mentioned in the examples. As a result, both the $\tilde{\lambda}_1$ (0.3%) and $\tilde{\lambda}_{10}$ (1.6%) in sentence choices from MneYanEx was less than those from MneSchEx, but is more than that from MnePerEx. The difference in the $\tilde{\lambda}_{10}$ between MneYanEx and MneGenEx was significant ($z = 5.91, p < 0.001$). 3 sentences were chosen twice.

Finding 4: Both personalized sentences and high-quality examples are needed to achieve better security. Another question is whether the instructions or the examples have more influence on the unpredictability of the chosen sentences and consequently the generated passwords. This led us to study the two variants MnePer and MneEx. MnePer asked for personalized sentences in instructions, but did not provide any example; while MneEx did not explicitly ask for personalized sentence in the instructions, but provided a list of personal sentences as examples. For MnePer, the most popular one was chosen 5 times ($\tilde{\lambda}_1 = 0.7\%$), and $\tilde{\lambda}_{10}$ was 2.8%. $\tilde{\lambda}_{10}$ from MnePer was significantly smaller than that from MneGenEx ($z = 4.42, p < 0.001$), and was significantly larger than that from MnePerEx ($z = 2.11, p = 0.017$). For MneEx, the most popular one was chosen 6 times ($\tilde{\lambda}_{10} = 0.7\%$), and $\tilde{\lambda}_{10}$ was 2.2%. $\tilde{\lambda}_{10}$ from MnePer was significantly smaller than that from MneGenEx

($z = 5.41, p < 0.001$), and was larger than that from MnePerEx ($z = 1.39, p = 0.083$). There was no significant difference between the MnePer and MneEx.

An unexpected finding is that in MnePer, while the 10 most popular sentences were chosen only 19 times, $\tilde{\lambda}_{10}$ in password choices was 5.6% ($\mathsf{top}_{10} = 42$). In all other strategies, the $\tilde{\lambda}_{10}$ in password selections was less than that in sentence selections, since the same sentence can result in different passwords. Why do we have higher frequency in popular passwords than in popular sentences? Examining the dataset we found that a significant fraction of users choose pure digit sequence passwords (such as 12345678, 233425233525) that did not appear to match the sentences. (Since we allow letters to be replaced with digits, we did not check for such situations.) It appears that when users are not shown any examples, some users do not know how to follow the instruction.

Overal, our results suggest that neither explicit request for personalized sentences nor high-quality examples by itself suffice (in fact, neither appears to be more important than the other), and one needs both to get high security.

Table 5.6.: Potential popular and personalized sentence counts evaluated by Google search results.

Strategy	MneGenEx	MnePerEx	MnePer	MneEx	MneSchEx	MneYanEx
Popular	242(28%)	13(1.7%)	42(5.6%)	27(3.1%)	89(12%)	126(16%)
Personal	521(60%)	757(97%)	680(91%)	825(95%)	635(84%)	644(81%)

Sentence Popularity. We further examined the popularity of the sentences by searching them on Google using exact matching, and obtain the estimated number of results. Ideally, a "true" personalized sentence should have 0 result returned. On the other hand, famous quotes and popular sentences yielded a high number of results. For example, the 5 sentences with highest estimated result numbers are "to be or not to be", "the quick brown fox jumps over the lazy dog", "when you wish upon a star", "do, or do not, there is no try", and "you can't always get what you want", which are all famous quotes. We say that a sentence is popular if Google's estimated result number is more than 100, and that a sentence is personalized if the estimated result number if less than 5. Table 5.6 shows the numbers of popular and personalized sentences for all mnemonic-based strategies. From the table, we can observe that the rates of popular sentences were reduced dramatically from 28% to around 4% when either personalization instructions or examples were introduced, and the combination of them further reduced the rate to 1.7%. This latter finding confirms the necessity of combining explicit request for personalization sentences and high-quality examples.

5.3.3 Cracking Mnemonic Passwords

We now develop a method for cracking passwords generated using the mnemonic strategy. Our goal is to demonstrate that the step of converting sentences to passwords provides only limited extra entropy. Given the sentences, we can crack more than half of the passwords selected by the users in between 5 and 10 guesses.

For ease of exposition, we first explain our method for case-insensitive passwords. When generating passwords by following the mnemonic strategy, a word can theoretically be mapped to any character; however, given a word, the number of characters that are chosen by the users is limited in practice. People generally just pick the first letter of each word. When ignoring case differences, on average, each word is converted to 4 possible characters.

Table 5.7 lists the character usage in passwords generated in mnemonic strategies. In the table, *Upper*, *Lower*, *Digit* and *Symbol* means the number of corresponding type of character used in passwords. *First* means the number of words whose first character is directly used in the password. *First + Leet* means the number of words whose first character or the Leet substitution of the first character is used in the password. *Total Trans* means the total number of pairs of word and resulted characters. *Unique Trans* means the number of word-character pairs that only appear once in the dataset. *Distinct Words* are the number of distinct words used in all sentences.

From Table 5.7, we can see that on average 81.2% of the words are converted into their first character; furthermore, about 3.3% of the time, an additional leet substitution is applied. For mappings not using the first letters, the characters chosen are almost fixed for a given word; most of them are based on pronunciation or the meaning of the word. For instance, "*to*" is mapped to "2", "*question*" is mapped to "?", and "*first*" is mapped to "1".

Given a training dataset which contains pairs of sentences and passwords, we first learn the probability distribution of the word-to-character mappings. We classify words into *normal words* and *special words*. Normal words are typically mapped to the first character, with a possible leet substitution. For each letter, we maintain a probability distribution of how that letter is likely to mapped into. Special words are often not mapped to its first letter. For each special word, we maintain a probability distribution for its mappings.

The classification of words is an iterative process. At the beginning, we assume that all words which appear at least 5 times are normal words. In each iteration, we first calculate the probability distribution of each character by averaging the converted character distribution of all corresponding words. Then, we find the `L1` distance between the converted character distribution of each word and the probability distribution of its first character. If the `L1` distance is larger than a certain threshold, we say that the word is a special word. In our experiment, the threshold value we use is 0.6. We repeat the process until no words are removed from normal words.

For password cracking, given a sentence, we first generate a guess by taking the first character of all the words. Then, we generate the passwords by converting words into characters. We assume that in a sentence, the same words are always converted in the same way, and different words are converted into characters independently. Therefore, the probability of each generated password is the product of the probabilities of the transitions from all unique words to characters. We generate passwords in the descending order of probability.

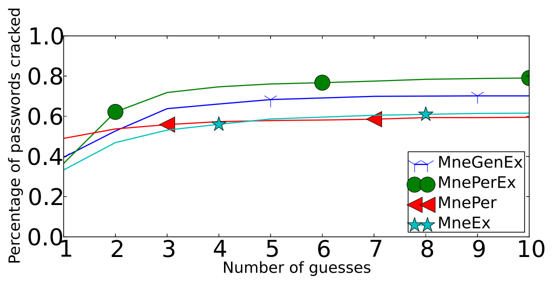
We evaluate the method on the sentences and passwords we collected from MneGenEx, MnePerEx, MnePer, MneEx by cross validation, i.e., we train the model on data from three strategies and attempt to crack passwords in the other strategy. MneSchEx and MneYanEx are excluded in the evaluation, as in the two strategies, a word is not always converted into one character. The percentage of passwords cracked when varying the number of guesses

Table 5.7.: Character usage in mnemonic strategies.

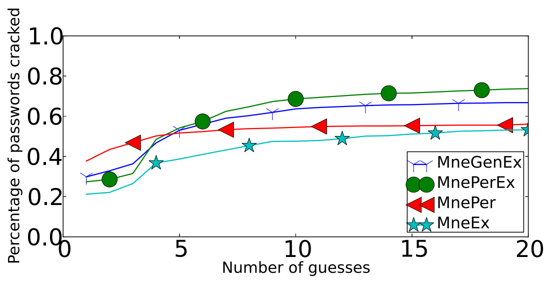
Strategy	Upper	Lower	Digit	Symbol	First	First+Leet	Total Trans	Unique Trans	Distinct Words
MneGenEx	683$_{8.6\%}$	6073$_{76.4\%}$	792$_{10.0\%}$	399$_{5.0\%}$	6633$_{83.5\%}$	194$_{2.4\%}$	7947	1758$_{22.1\%}$	2034
MnePerEx	559$_{7.8\%}$	5465$_{76.7\%}$	741$_{10.4\%}$	364$_{5.1\%}$	6080$_{85.3\%}$	174$_{2.4\%}$	7129	1718$_{24.1\%}$	1954
MnePer	817$_{12.0\%}$	4290$_{63.2\%}$	1123$_{16.6\%}$	554$_{8.2\%}$	5027$_{74.1\%}$	332$_{4.9\%}$	6784	2228$_{32.8\%}$	2334
MneEx	994$_{12.2\%}$	5539$_{68.1\%}$	1046$_{12.9\%}$	553$_{6.8\%}$	6651$_{81.8\%}$	277$_{3.4\%}$	8132	2547$_{31.3\%}$	2207

is illustrated in Figure 5.2(a). For all of the strategies, we can crack 60% of the passwords within 10 guesses, where most of them are in the first 5 attempts.

The method performs less effective on MnePer and MneEx. From Table 5.7, we can observe that the percentages of unique conversions from a word to a character contribute to 32.8% and 31.3% of all such conversions in MnePer and MneEx. The quantities are much higher than those from MneGenEx (22.1%) and MnePerEx (24.1%). The more unique conversions leads to more character mappings that are never guessed. Table 5.7 also shows that participants in MneEx and MnePer are more likely to use digits, symbols, and upper-case letters than participants in MneGenEx and MnePerEx. One likely explanation is that just a single example is presented in MneGenEx and MnePerEx; while no example is presented in MnePer and six examples are given for MneEx. It is possible that both no example and lots of examples cause people to be more creative in mapping words to characters.



(a) case-insensitive



(b) case-sensitive

Figure 5.2.: Percentage of passwords cracked within 10 attempts for case-insensitive passwords, and 20 attempts for case-sensitive passwords.

We adapt our method to be case-sensitive when guessing as follows. The training process is identical to the case-insensitive condition. When generating password guesses, every time a password (with the highest probability) is generated, instead of 1 guess, 4 guesses are made. We try the original password, capitalize all letters, capitalize the first letter, and capitalize all letters whose corresponding words are capitalized. The performance of the method on case-sensitive passwords is shown in Figure 5.2(b). More than 50% of passwords in all the 4 strategies can be guessed in 20 attempts, with most successes from the first 10 guesses.

Crack from Scratch. Now, we apply the cracking method described above to a real-world scenario, in which sentence selection in the testing dataset is unknown. Given a training dataset, we generate candidate passwords as follows. We first order the sentences selected in the training dataset by the descending order of their frequencies. Then, starting from the most popular sentence, for each sentence, we generate 20 case-sensitive guesses.

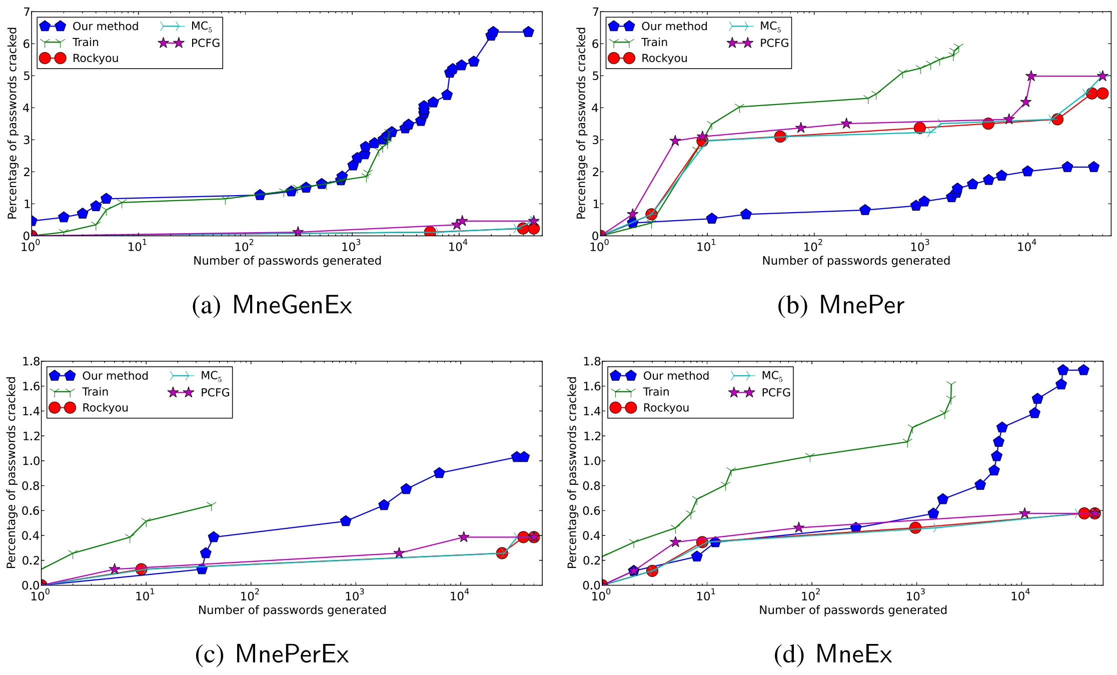


Figure 5.3.: Guess number graph on passwords created by using mnemonic strategies.

Fig 5.3 shows the effect of our method evaluated on the four datasets by cross validation, i.e., for each testing dataset, the training dataset is the union of the other three datasets. In the graphs, each curve represents a cracking method, and a point (x, y) on the curve means y percentage of passwords in the testing dataset are cracked within x attempts. We also plot the curves of 5-order Markov Model (MC_5), PCFG method (PCFG) trained on Rockyou dataset, and two blacklist-based methods, which use Rockyou (Rockyou) dataset and the passwords in the training datasets (Train), respectively. Because of the limited number of sentences in the training dataset (less than 2400), we are able to generate less than 50,000 candidate passwords using the new method, and used 50,000 as the number of passwords generated for all methods.

The evaluation on MneGenEx is illustrated in Fig 5.3(a). From the figure, we can observe that all the generic cracking methods perform poorly, and can crack no more than 0.4% passwords within 50,000 guesses. In fact, the only passwords covered by the methods are "!@#$%^" and "!@#$%^&*", which are apparently created without following the strategy. On the other hand, 3.2% of the passwords in MneGenEx are covered in the 211 passwords in the training datasets, which confirm the need to using strategy-specific methods. Our proposed method can crack 6.4% passwords with 50,000 guesses. We expect performance of the method will increase with the size of training data. The performance of our method as well as the dictionary obtained from the training dataset drops significantly on the other datasets, and passwords from MnePerEx appears to the strongest. Less than 1% passwords in MnePerEx are cracked with 50,000 guesses. The reduced performance of the methods is mainly due to the requirement of personalized sentence choice and the resulted increasing number of unique sentences. The result is consistent with the findings from $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ analysis described in Section 5.3.2.

One may also notice that in MnePer, the relative order of the methods is quite different. This is because of high frequency of passwords generated not following the strategy, such as *12345678* (17), *!@#$%^&** (6), *123456789* (5). These passwords are hard to predict based on our method, but are easy to guess based on the other methods. As a result, our method performs worse than all the other methods in the graph.

5.4 Study 2: Usability

We conducted another user study evaluating the usability of the mnemonic strategies from two aspects: (I) Creation usability: Time and effort required from the user to generate a password by following the given strategy; (II) Memorability: long-term retention of the password generated with the given strategy. In this study, three strategies were evaluated, MneGenEx, MnePerEx, which are evaluated as the most and the least secure mnemonic strategy in the previous analysis, and Control, which serves as a baseline.

Time used for password generation, the success rate of password recall, and password recall time were measured. We also examined the effort that participants spent during password generation and retention utilizing the NASA-Task Load Index (TLX) [66], in which the workload is rated on 6 subscales: mental demand, physical demand, temporal demand, performance, effort and frustration. In the NASA-TLX, participants rated the workload of each subscale ranging on a linear scale from 0 to 20, where 0 means very low workload and 20 means high workload.

5.4.1 Study Overview

This study was conducted on MTurk in two phases. The first phase was similar to the previous experiment except as noted. At the beginning, we explicitly told participants that they would be asked to return and use the password in about one week, and they could take whatever measures they would normally take to remember and protect the passwords. Also, a concrete creation scenario was provided to simulate a real-world password generation context. Specifically, each participant was asked to create an online account for a bank named "Provident Citizens Bank". One strategy randomly selected from Control, MneGenEx, and MnePerEx was assigned to each participant for password generation. For participants using MneGenEx and MnePerEx, the sentence creation and password generation were separated into two pages, such that the created sentence was not visible to participants during password generation, in order to mimic the password generation environment in practice. Participants were allowed to arbitrarily switch back and forth between the two pages.After the password generation, each participant was asked to measure the workload spent on creating the passwords utilizing the NASA-Task Load Index. About half of the participants were randomly selected to recall the password that they had just created at the end of the study, to evaluate the impact of short-term rehearsal on long-term password recall.

Participants were invited back for the second phase by email. We sent the invitation emails through MTurk starting from the 6th day after participants finished the first phase. For those participants who did not come back to the study, we re-sent the same invitation email for another two days. In the second phase, participants were instructed to login to "Provident Citizens Bank" with the password they created and then to update the password. Each participant was allowed up to 4 attempts until failure. If a participant could not recall the password within the first 2 attempts, the strategy was displayed as a hint. Regardless of the performance in the login process, all participants were asked to evaluate the workload during password recall by using the NASA-TLX afterwards.

Table 5.8 and Table 5.9 list the general statistics of the first-phase and second-phase study, respectively. In the first phase, for each condition, we list the number of participants, average password creation time, and statistics for short-term password recall (if applicable) including success rate before and after seeing the strategy as a hint, failure rate after 4 attempts, and time used in password recall. In the second phase, we list the number (percentage) of participants that returned to the study; statistics about long-term password recall, including the number (percentage) of participants who did not write down passwords; the success/failure rate and average time used in password recall for those who did not write passwords down; the number (percentage) of participants who used the strategy provided to update their passwords. In the table, *Succ1* means the the number of participants who successfully recall the password within 2 attempts. *Succ2* means the the number of participants who successfully recall the password in the third or fourth attempts, and the strategy was displayed as a hint. *No WDP* means the number of participants who did not write down passwords. Time is measured in seconds.

5.4.2 First Phase Results

We recruited 425, 400, and 435 participants for Control, MneGenEx, and MnePerEx, accordingly, with a total of 1260 (614 females). The participants' ages ranged from 18 to over 50, with 76% between 23 to 50 years. Most participants were college students or professionals who had bachelor or higher degrees.

Table 5.8.: Statistics for first-phase usability study for mnemonic strategies.

Strategy	Short Term Recall	Phase 1					
		Count	Creation Time	Short-term Recall			
				Succ1	Succ2	Failed	Time
Control	Yes	217	38.7	213(98%)	1(0%)	3(1%)	26.0
	No	208	39.8	N/A	N/A	N/A	N/A
	All	425	39.2	N/A	N/A	N/A	N/A
MneGenEx	Yes	185	177.2	178(96%)	5(3%)	2(1%)	30.6
	No	215	146.7	N/A	N/A	N/A	N/A
	All	400	160.9	N/A	N/A	N/A	N/A
MnePerEx	Yes	221	133.3	212(96%)	3(1%)	6(3%)	30.4
	No	214	149.5	N/A	N/A	N/A	N/A
	All	435	141.3	N/A	N/A	N/A	N/A

Password Creation Time. The average time used in password creation for each strategy is listed in Table 5.8. The password creation time was significantly different among the three strategies. As expected, participants spent the least time when there was no restriction (Control), and time spent in Control is significantly less than that MneGenEx and MnePerEx ($p < 0.001$). Compared with MneGenEx, password generation time was shorter in MnePerEx ($t = 2.14, p = 0.033$). That's mainly due to the additional requirement for personalized choice that narrowed down the search space of sentences and resulted in a faster decision.

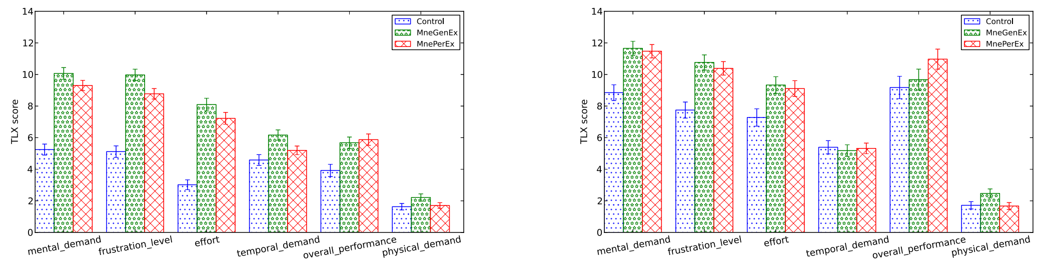
Workload. Fig 5.4(a) shows the average ratings in each subscale of NASA-TLX for the three strategies. Overall, the perceived workload was relatively low, with the average ratings for all subscales being below or close to 10. The workload required in Control were lower than that from the two mnemonic strategies. The workload required in MneGenEx

Table 5.9.: Statistics for second-phase usability study for mnemonic strategies.

Strategy	Short Term Recall	Phase 2						
		Number Returned	Long-term Recall					Update
			No WDP	Succ1	Succ2	Failed	Time	Use Strategy
Control	Yes	169(78%)	140(83%)	51(36%)	9(6%)	80(57%)	43.6	N/A
	No	154(74%)	125(81%)	48(38%)	9(7%)	68(54%)	48.6	N/A
	All	323(76%)	265(82%)	99(37%)	18(7%)	148(56%)	46.0	N/A
MneGenEx	Yes	156(84%)	129(83%)	62(48%)	10(8%)	57(44%)	70.1	105(67%)
	No	166(77%)	142(86%)	47(33%)	13(9%)	82(58%)	113.1	113(68%)
	All	322(80%)	271(84%)	109(40%)	23(8%)	139(51%)	92.6	218(68%)
MnePerEx	Yes	180(81%)	146(81%)	44(30%)	8(5%)	94(64%)	112.9	124(69%)
	No	161(75%)	135(84%)	38(28%)	9(7%)	88(65%)	103.5	116(72%)
	All	341(78%)	281(82%)	82(29%)	17(6%)	182(65%)	108.4	240(70%)

and MnePerEx is similar, and MnePerEx results in a lower workload in most of the subscales. This is consistent with the comparison in password creation time.

Short-term Memorability. About half of the participants in each strategy were asked to recall the password at the end of the first phase. From Table 5.8, we can observe that regardless of the strategy used, almost all participants could enter the correct password.



(a) Password Generation (b) Password Recall

Figure 5.4.: Mean scores of TLX as a function of strategy and subscale for Control, MneGenEx, and MnePerEx. Error bars represent standard errors of the scores.

5.4.3 Second Phase Results

Approximately 78% of participants from each strategy condition returned to the second phase of the study.

Long-term Memorability. For participants who came back for the study, approximately 83% indicated that they did not write down the password or the sentence (in MneGenEx and MnePerEx), and the ratio for the three strategies is similar, indicating that participants using mnemonic stratetics were as confident as those in the control group, that they could remember the passwords. We analyzed the password memorability from the participants who claimed that they did not write down passwords or sentences.

The ratio of participants recalled the passwords successfully within first two attempts range from 29% to 40%, depending on the strategy used, and an extra 7% of participants were able to recall the passwords when the strategy was displayed as a hint. The performance of Control and MneGenEx are similar ($\chi^2 = 1.31, p = 0.521$). Both Control

($\chi^2 = 4.66, p = 0.097$) and MneGenEx ($\chi^2 = 10.31, p = 0.006$) perform significantly better than MnePerEx, indicating that the long-term memorability of MnePerEx is relatively bad. When there was a short-term recall, the long-term recall success rates were generally increased for each strategy, and the increase in rates was larger for the mnemonic strategies, especially for the MneGenEx strategy.

Long-term Recall Time. The password recall time in Control was shorter than that for MneGenEx ($t = 4.64, p < 0.001$) or MnePerEx ($t = 2.82, p = 0.005$), whereas there was no significant difference between MneGenEx and MnePerEx ($t = 0.67, p = 0.505$). Again, whether or not short-term recall had been required did not have any significant impacts.

Workload. The workload of password retention evaluated by TLX is illustrated in Fig 5.4(b). Comparing Fig 5.4(a) and Fig 5.4(b), perhaps the most noticeable difference is that the subscale of performance in Fig 5.4(b) is almost double that in Fig 5.4(a), which was mainly due to large portion of failed recall. Except physical demand and temporal demand, which are not directly related to the task, the average rates of all the other 3 subscales also increased dramatically, suggesting that password recall was more difficult than password generation.

For the subscales, mental workload and frustration ratings of mnemonics strategies were higher than those of the Control, and the rating for MneGenEx is slightly higher than MnePerEx, which is consistent with the first phase results, suggesting the overhead from the extra requirements of the mnemonic strategies and MnePerEx is considered to be easier to use than MneGenEx.

Password Update. At the end the task, we asked participants to update the password, without any restriction except that the password could not be the same as the old one. For MneGenEx and MnePerEx, after the password was created, we asked participants whether they used the strategy we provided. About 70% of participants said "yes" to the question, and the percentage for MnePerEx was slightly higher. The results indicated that most of the participants were willing to use the instructed strategy even if not forced to do so.

Overall, the study suggests that although workload required for MneGenEx are significantly larger than that for Control, no significant difference in password memorability between the two strategies is observed, which is consistent with the previous literature [46, 47]. However, MnePerEx, which shows advantage over MneGenEx in terms of security, has a quite different behavior. It is considered to be easier to use than MneGenEx, with the cost of higher failure rate in long-term memorability.

5.5 Discussion and Conclusion

We conducted our study with the MTurk population, which is more diverse than the participants in typical laboratory studies [67]. The use of MTurk also allowed us to recruit more participants and collect data with larger samples, which cannot be easily done in traditional in-person data collection. We found that passwords created in the control group (Control) is stronger than that of the existing real-world password datasets, such as Rockyou. Therefore, the security of passwords created in the study can serve as a lower-bound measurement.

In this chapter, we investigated the security and usability of 6 variants of mnemonic strategies. Our studies improved the understanding of password generation strategies through the following contributions.

- We showed that using the standard cracking-based methodology, password sets obtained under all variants had similar strengths and were all much more secure than the baseline. However, using β-guess-rates, we found that using generic instructions that had been suggested in the literature resulted in 2.5% of the group choosing the same sentence, and the top 10 sentences chosen by 7.8% of the group. We have also found that converting a sentence to a password added very limited entropy. These two facts together suggested that this variant of mnemonic strategy is no more secure than the baseline.
- We showed that combining explicit instruction of choosing a personalized sentence that is unlikely to be chosen by others, with the inclusion of such personalized ex-

amples, dramatically increased the security of the resulting passwords. Furthermore, using only the explicit instruction or the examples alone resulted in less secure distributions.

- We showed that the instructions for the mnemonic strategy found in the literature and recommended by security experts were not optimal in inducing secure password distributions.

- We found that requiring personalized choice of sentences in mnemonic strategies effected the usability of the mnemonic strategy. With the additional requirement, the strategy turned out to be easier to use in terms of password creation. However, the performance of long-term memorability decreased.

6. UNDERSTANDING WORD-BASED PASSWORD GENERATION STRATEGIES

In this chapter, we evaluate variants of another commonly suggested password generation strategy made famous by the xkcd comic [25]. In the xkcd comics, one is suggested to choose 4 random words and concatenate the words to make a password. Similar strategies were also recommended by the security community. For example, combining two or three unrelated words and changing some of the letters to numbers or symbols [24]; mixing letters from two words was suggested in [31], etc.

Observing that password creation steps in these strategies are similar: first pick 2 to 4 words, then somehow combine the words into a password; and passwords are generally constructed based on words, we categorize these strategies as word-based password strategies. In this chapter, we analyze the security and usability of 7 variants of word-based strategies in a series of human subject studies which are similar to those described in Chapter 5.

6.1 Introduction

Beyonds mnemonic sentence-based strategies, word-based strategies are commonly recommended [24, 25, 31] as well. In such strategies, a password is generally created in a two-step process: first choose some unrelated or random words (usually 2 to 4), then combine the words into a password.

Despite the popularity of the strategies, to the best of our knowledge, the strategies have not been evaluated in any human subject studies and little is known about the effectiveness of them in helping users create usable and secure passwords.

In this chapter, we examine the security and usability of 7 variants of the strategy in a series of studies. The security of the variants was evaluated in a study conducted on MTurk in which a total number of $5,484$ participants were involved. In the study, each participant was asked to create 1 or 2 passwords following different variants of the strategy. When assessing the security of the variants, we adopt the approach of using statistical quantities to measure the distributions of the passwords, as articulated by Bonneau [28]. In particular, we chose to use the *β-guess-rate* (λ_β) [29], which measures the expected success for an attacker limited to β guesses per account. We chose to use $\beta = 1$ and $\beta = 10$, as they were suggested in [30] as appropriate for defense against online guessing attacks and because a larger β is not very meaningful for our sample sizes.

The usability of the variants is evaluated in a separate user study, in which password creation time, short-term and long-term password memorability, and the workload required in both password creation and retention are evaluated.

Our studies were conducted on Amazon Mechanical Turk, and the studies were similar to those described in Chapter 5.

6.2 Study 1: Security

In this section, we present an overview of the first study.

We study 7 variants of word-based strategies. In such a strategy, a participant is asked to first select a few (2, 3, or 4) unrelated words, and then combine (e.g., through concatenation, interleaving of the words, connected with punctuation) them to obtain the password. For instance, if the user is asked to select 3 words, and selects the words, "*correct*", "*horse*", and "*stable*", then one possible password according to this strategy can be "*correcthorsestable*".

Table 6.1.: Word-based strategy description

Strategy	Short Description	Exact instruction given to the users in the study
4WordEx	4 unrelated words [25]	1. Choose 4 unrelated words that make sense to you. For example, “correct, horse, battery, staple”. 2. Combine them in order into one string and use that string as a password: “correcthorsebatterystaple”.
3WordEx	3 unrelated words with example	1. Choose 3 unrelated words that make sense to you. For example, “correct, horse, battery”. 2. Combine them in order into one string and use that string as a password: “correcthorsebattery”.
3Word	3 unrelated words	1. Choose 3 unrelated words that make sense to you. 2. Combine them in order into one string and use that string as a password.
3or2WordEx	2/3 unrelated words concatenated together and some letters tweaked [24]	1. Choose two or three unrelated words. 2. Combine the words and change some of the letters to numbers or special characters. Examples: “bank” and “camera” might become “B@nkC@mera”. “mail” and “phone” might become “m4!lf0N3”.

continued on next page

Table 6.1.: *continued*

Strategy	Short Description	Exact instruction given to the users in the study
2WordMixEx	Two words mixed [31]	1. Choose two unrelated words of approximately the same length. For example, “house” and “plane”. 2. Interleave the two words (Start from the first letter of the first word and the first letter of the second word, and repeat this until you get to the last letter of each word) to create the password. For example, “hpoluasnee”.
2WordPunEx	Two words connected by punctuation.	1. Choose two unrelated short words. For example, “house” and “plane”. 2. Concatenate them together with a punctuation character between them. For example, “house+plane”.
2WordPun	Two words without example.	1. Choose two unrelated short words. 2. Concatenate them together with a punctuation character between them.

Table 6.1 gives details of the 7 variants in our study. We urge readers to read the table before proceeding, as the differences in the strategy descriptions are an important part of the study. Below is a summary.

- 4WordEx (4-Words-Example, 4 unrelated words, with the xkcd example [25]),
- 2WordMixEx (2-Words-Mix-Example, interleaving 2 words, with an example, used in [31]),
- 2WordPunEx (2-Words-Punctuation-Example, two words with a punctuation connecting them, with an example),

- 3WordEx (3-Words-Example, 3 unrelated words, with an example),
- 3or2WordEx (2-or-3-Words-Example, 2 or 3 unrelated words with some letters replaced with special symbols or digits, with examples, suggested in [24]),
- 3Word (3-Words, 3 unrelated words, no example),
- 2WordPun (2-Words-Punctuation, two words with a punctuation, no example).

In addition to the 7 variants, a control group Control, in which we ask for passwords containing at least 8 characters without any extra restriction, was included in the study as well.

Table 6.2.: Word-based strategies used in each round of the study.

Study	Number	Strategies
1	864	4WordEx, 2WordMixEx
2	777	4WordEx, 2WordPunEx
3	753	3WordEx, 3or2WordEx
4	745	3Word, 2WordPun
5	868	4WordEx
6	799	4WordEx
7	678	Control

6.2.1 Study Design

We conducted the study through Amazon Mechanical Turk (MTurk), and all participants were at least 18 years old. We limited our data collection to participants from the United State because the strategies were constructed using the English language. The study has 7 rounds. Except the control group (Control in Round 7), in each round, participants were asked to create passwords for one or two variants of word-based strategy. The variants presented in each round as well as the number of participants recruited are listed in Table 6.2. If multiple variants were studied in one round, the order of the variants presented was randomized among participants. Participants were allowed to participate in only one round. If a participant was in more than one rounds of the study, we kept only the data from the first time that the participant was in.

In the study, participants were asked to type the words used in the intermediate step. After that, participants were asked to enter the password they created twice, and the password typed in each time had to match each other before they could proceed.

We warned participants not to use their actual passwords. In the study, we forbade passwords that were the same as the examples and that did not appear to be generated following the instructions. In all the variants of word-based strategies except 3or2WordEx, once the words were chosen, the sequence of letters in the passwords was fixed. We checked if the password one generated matched with the words. In 3or2WordEx, since we asked participants to change some letters to numbers or special symbols, and participants were allowed to, e.g., replace syllables with letters with similar pronunciation (e.g., "ph"$\rightarrow$"f"), we only required that there were non-letter characters in the passwords.

6.3 Results From Study 1

We evaluate the security of passwords as well as words used in password creation utilizing the same metrics described in Chapter 5.

6.3.1 Analyzing Passwords Using Probability Models and Password Strength Meters.

Figure 6.1 illustrates the evaluation of the strength of passwords generated using the strategies as well as two commonly used datasets Yahoo and Phpbb against today's attacks utilizing (1) the 5-order Markov Model trained on Rockyou dataset, (2) Google password strength API[1], and (3) Zxcvbn [23] deployed by Dropbox.

In all the three graphs, the curve for the control group (Control) is below the curves for Yahoo and Phpbb, indicating that passwords created in the study are stronger than that in real-world datasets. Therefore, the security of passwords created in the study can serve as

[1]https://accounts.google.com/RatePassword

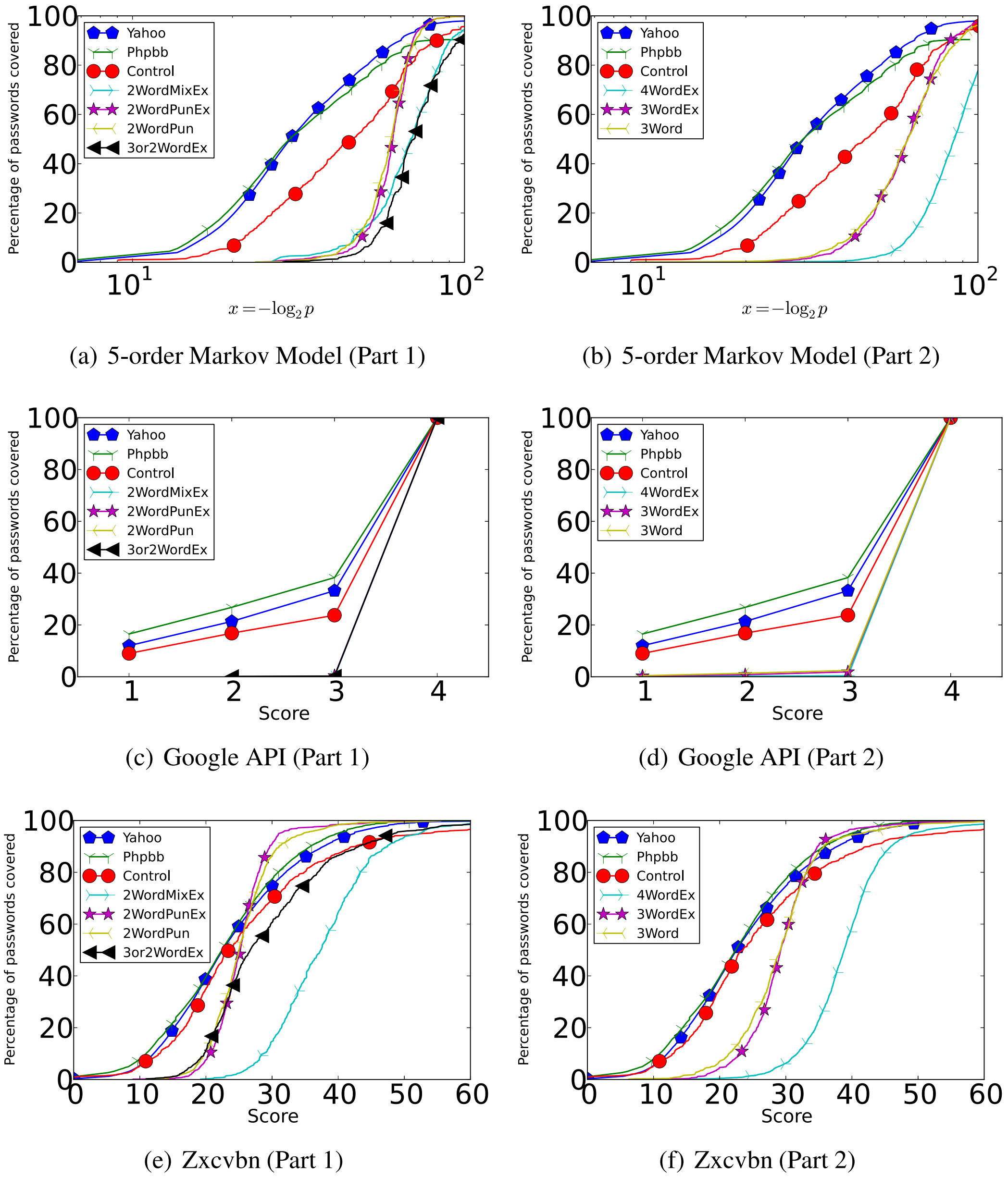


Figure 6.1.: Comparison of strength of passwords from different word-based strategies and datasets using probabilistic models and password strength meters.

a lower-bound measurement. On the other hand, curves for Yahoo, Phpbb, and Control are significantly higher than the other curves in most of the graphs. The only exception is that Figure 6.1(e), the performance of 2WordPunEx and 2WordPun is relatively good. This is because in *zxcvbn*, the entropy of passwords created in the two strategies are the sum of the entropy of the two selected words and the entropy of the punctuation. Since the entropy of a single charater (the punctuation) is limited, the estimated entropy in *zxcvbn* is approximately the entropy of two words. The comparison between Yahoo, Phpbb, Control and the other strategies indicates that according to the metrics, passwords created without any specific strategy are significantly weaker than the other datasets. When guessing according to the order suggested by the metrics, more passwords in Yahoo, phpbb, and Control will be cracked than passwords from any word-based strategy. However, this conclusion is due to the fact that the model or the meters are designed to evaluate generally selected passwords, which is broadly similar to passwords in Yahoo and Phpbb, and passwords generated from the word-based strategies result in quite different distributions.

Among the word-based strategies, the variants requiring more words are evaluated as stronger in general. The only exception is 2WordMixEx. Based on Markov Model and *zxcvbn*, 2WordMixEx performs similar to 4WordEx in terms of security. After interleaving letters, the resulted passwords are likely to be considered as random strings by the models and password meters, which results in high entropy assigned.

6.3.2 Analyzing Passwords Using Statistical Quantities.

We evaluate the strength of words used in strategies as well as the resulted passwords utilizing $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ metrics. The evaluation of Control is listed in Table 5.3 and has been discussed in Chapter 5. Table 6.3 gives the $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top$_{10}$) values of sentences and the resulted passwords for all mnemonic sentence-based variants.

When counting the collisions in word selections, the order of the words is not taken into account, i.e., we first sort the chosen words and then evaluated on the sorted words. In 2WordMixEx, the minimum length requirement of passwords is 6, while the other variants

Table 6.3.: $\tilde{\lambda}_1$ (top) and $\tilde{\lambda}_{10}$ (top_{10}) in word-based strategies.

Strategy	Count	$\tilde{\lambda}_1$ (top)			$\tilde{\lambda}_{10}$ (top_{10})		
		Words	Password		Words	Password	
			Case Insensitive	Case Sensitive		Case Insensitive	Case Sensitive
3WordEx	753	0.1%(1)	0.1%(1)	0.1%(1)	1.3%(10)	1.3%(10)	1.3%(10)
3Word	745	0.3%(2)	0.3%(2)	0.3%(2)	1.6%(12)	1.6%(12)	1.5%(11)
3or2WordEx	753	0.3%(2)	0.1%(1)	0.1%(1)	2.1%(16)	1.3%(10)	1.3%(10)
2WordMixEx	864	3.8%(33)	2.0%(17)	2.0%(17)	6.8%(59)	6.1%(53)	6.0%(52)
$\mathsf{2WordMixEx}_8$	761	0.5%(4)	0.5%(4)	0.5%(4)	3.7%(28)	3.2%(24)	3.0%(23)
2WordPunEx	777	0.4%(3)	0.3%(2)	0.3%(2)	3.2%(25)	1.5%(12)	1.5%(12)
2WordPun	745	0.4%(3)	0.1%(1)	0.1%(1)	2.6%(19)	1.3%(10)	1.3%(10)
4WordEx	3307	0.1%(2)	0.1%(2)	0.1%(2)	0.5%(15)	0.4%(14)	0.4%(13)
3WordAll	1709	0.2%(3)	N/A	N/A	1.1%(19)	N/A	N/A
2WordAll	2927	1.2%(34)	N/A	N/A	2.7%(79)	N/A	N/A
Control	678	N/A	1.2%(8)	0.9%(6)	N/A	3.4%(23)	2.9%(20)

all require passwords with at least 8 characters. If we remove all passwords whose length is less than 8 in 2WordMixEx, we are left with 761 passwords. We label this set of passwords as 2WordMixEx$_8$.

We observe that there is no significant differences in terms of $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ for variants requiring the same number of words. We find that the most frequently used words in each variant and present the results in Table 6.4. The table suggests that the most popular words are almost identical in all the variants.

Table 6.5.: Popular passwords and probability for top 5 frequently chosen word combinations in word-based strategies.

Rank	Words	Passwords	Frequency
4WordEx (3307)			
1	ate cat dog food (2)	dogatecatfood (2);	0.06%
2	brown fox quick the (2)	thequickbrownfox (2);	0.06%
3	four one three two (2)	onetwothreefour (1); ONETWOTHREEFOUR (1);	0.06%
4	dog fox lazy quick (2)	quickfoxlazydog (1); quickdoglazyfox (1);	0.06%
5	blue green red yellow (2)	redyellowgreenblue (1); redbluegreenyellow (1) ;	0.06%
3WordEx (753) No collisions found.			
3Word (745)			
1	battery horse staple (2)	horsebatterystaple (2);	0.27%
2	i love you (2)	Iloveyou (1); iloveyou (1);	0.27%
3or2WordEx (753)			
1	blanket dog (2)	d@gblank*t (1); d@gb1anket (1);	0.27%
2	cat school (2)	c4t$c#00l (1); c4t$ch00l (1);	0.27%
3	car house (2)	H0us3c@r (1); h0us3ca7 (1);	0.27%

continued on next page

Table 6.5.: *continued*

Rank	Words	Passwords	Frequency
4	dog phone (2)	p40ned0g (1); d09ph0n3 (1);	0.27%
5	fuck you (2)	Fuck&you (1); f@#$ y@u (1);	0.27%
2WordMixEx (864)			
1	cat dog (33)	cdaotg (17); dcoagt (16);	3.8%
2	chair table (4)	tcahbalier (3); cthaabilre (1);	0.46%
3	hate love (4)	lhoavtee (4);	0.46%
4	house mouse (4)	hmoouussee (2); mhoouussee (1); MHOOUUSSEE (1);	0.46%
5	chair couch (3)	ccohuacihr (2);	0.35%
2WordPunEx (777)			
1	dog tree (3)	tree&dog (1); dog!tree (1); dog+tree (1);	0.39%
2	car tree (3)	tree:car (1); tree-car (1); car!tree (1);	0.39%
3	house tree (3)	tree-house (1); tree&house (1); tree!house (1);	0.39%
4	car horse (3)	horse!car (2); horse+car (1);	0.39%
5	cat chair (3)	cat!chair (1); chair+cat (1); cat+chair (1);	0.39%
2WordPun (745)			
1	hate love (3)	love;hate (1); love*hate (1); love$hate (1);	0.40%
2	ninja tiger (2)	ninja@tiger (1); tiger_ninja (1);	0.27%
3	box house (2)	box.house (1); box!house (1);	0.27%
4	dog flying (2)	dog.flying (1); dog!flying (1);	0.27%

continued on next page

Table 6.5.: *continued*

Rank	Words	Passwords	Frequency
5	cat computer (2)	cat@computer (1); Cat.Computer (1);	0.27%
3WordAll (1709)			
1	battery horse staple (3)	N/A	0.18%
2	i love you (2)	N/A	0.12%
3	car dog house (2)	N/A	0.12%
4	bat corn dog (2)	N/A	0.12%
5	car dog poker (2)	N/A	0.12%
2WordAll (2927)			
1	cat dog (34)	N/A	1.2%
2	hate love (8)	N/A	0.27%
3	house mouse (6)	N/A	0.21%
4	cats dogs (6)	N/A	0.21%
5	dog phone (5)	N/A	0.17%

Two Words. When asked to choose two random words and merge their letters one by one (2WordMixEx), 19 pairs of words were selected more than once, for a total of 77, and the 10 most popular pairs of words (top_{10}) were picked 59 times. 33 participants chose the words "*cat dog*". This yielded $\tilde{\lambda}_1 = 3.8\%$, $\tilde{\lambda}_{10} = 6.8\%$. If we remove the passwords with length less than 8 from 2WordMixEx, then $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ are reduced to 0.5% and 3.7%, as the most frequently appeared words are "*cat dog*".

Beyond "*cat dog*", the other frequently chosen word combinations were "*house mouse*", and "*hate love*". See Table 6.5 for other commonly chosen words, and the resulted passwords.

Although the instructions explicitly asked the users to choose two unrelated words in the description, people leaned towards choosing two words that were somewhat related to

Table 6.4.: Word frequency in word-based strategies.

Strategy	Total	W1	W2	W3	W4	W5
4WordEx	13504	dog ($380_{2.81\%}$)	cat ($266_{1.97\%}$)	car ($225_{1.67\%}$)	phone ($172_{1.27\%}$)	house ($160_{1.18\%}$)
3WordEx	2259	dog ($57_{2.52\%}$)	cat ($44_{1.95\%}$)	car ($39_{1.73\%}$)	house ($22_{0.97\%}$)	love ($20_{0.89\%}$)
3Word	2235	dog ($75_{3.36\%}$)	cat ($47_{2.10\%}$)	car ($37_{1.66\%}$)	love ($26_{1.16\%}$)	blue ($23_{1.03\%}$)
3or2WordEx	1717	dog ($50_{2.91\%}$)	cat ($39_{2.27\%}$)	car ($28_{1.63\%}$)	phone ($28_{1.63\%}$)	tree ($18_{1.05\%}$)
3or2WordEx $_{2w}$	1084	dog ($27_{2.49\%}$)	phone ($21_{1.94\%}$)	cat ($20_{1.85\%}$)	book ($12_{1.11\%}$)	chair ($10_{0.92\%}$)
3or2WordEx $_{3w}$	633	dog ($23_{3.63\%}$)	car ($20_{3.16\%}$)	cat ($19_{3.00\%}$)	tree ($10_{1.58\%}$)	phone ($7_{1.11\%}$)
2WordMixEx	1727	cat ($47_{2.72\%}$)	dog ($47_{2.72\%}$)	mouse ($32_{1.85\%}$)	phone ($22_{1.27\%}$)	book ($22_{1.27\%}$)
2WordPunEx	1554	dog ($65_{4.18\%}$)	cat ($49_{3.15\%}$)	car ($36_{2.32\%}$)	tree ($23_{1.48\%}$)	house ($21_{1.35\%}$)
2WordPun	1490	dog ($49_{3.29\%}$)	cat ($46_{3.09\%}$)	car ($19_{1.28\%}$)	apple ($13_{0.87\%}$)	house ($12_{0.81\%}$)
3WordAll	5127	dog ($155_{3.02\%}$)	cat ($110_{2.15\%}$)	car ($96_{1.87\%}$)	love ($51_{0.99\%}$)	blue ($44_{0.86\%}$)
2WordAll	5854	dog ($188_{3.21\%}$)	cat ($162_{2.77\%}$)	car ($76_{1.30\%}$)	phone ($67_{1.14\%}$)	mouse ($60_{1.02\%}$)

each other. This observation can be explained by Spreading Activation, which has been recognized in Cognitive Psychology [68]. A fairly widely accepted view is that memory is organized as a network of associations, and when one activates one item it will activate related ones.

For 2WordPunEx, 17 word combinations were chosen more than once, and $\tilde{\lambda}_{10}$ is 3.2%. For 2WordPun, 8 word combinations were chosen more than once, and $\tilde{\lambda}_{10}$ is 2.6%. The $\tilde{\lambda}_{10}$'s in password selections for 2WordPunEx and 2WordPun are much less than those in the word selections, this is primarily due to the choice of punctation. No significant difference is observed in $\tilde{\lambda}_{10}$ in word selections from the three two-word variants.

Combining all variants that require 2 words and those who chose only two words in 3or2WordEx, $2,927$ word pairs were collected, with 88 word combinations selected more than once, and $\tilde{\lambda}_1 = 1.2\%$, $\tilde{\lambda}_{10} = 2.7\%$. The quantities are similar to that from Control, and no significant difference is observed ($z = 0.98, p = 0.16$). Observing that both $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ from word combination selections are larger than those of password choices in Rockyou dataset, the security provided by 2-word based variants is far from satisfactory. Additional entropy (e.g., adding punctuation between words) can be introduced during the step where words are transformed into password but they are likely to be predictable. Table 6.6 lists the 5 most popular chosen punctuations in 2WordPunEx and 2WordPun as well as their frequency. From the table, we can observe that the top 5 punctuations count for more than 70% of all punctuation choices. If the word combination is known, more than 70% passwords will be cracked within 5 attempts.

Table 6.6.: Punctuation frequency in 2WordPunEx and 2WordPun.

Strategy	Total	P1	P2	P3	P4	P5
2WordPunEx	777	+(24%)	!(13%)	-(12%)	.(11%)	&(11%)
2WordPun	745	.(31%)	!(26%)	-(12%)	,(6%)	?(3%)

Three Words. For the two three-words variants, the only two word combinations selected more than once are "*battery horse staple*" and "*I love you*". The former is the first three words from the example in the xkcd comics. Not surprisingly, $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ from three-word based variants are all much lower than those from the two-word based variants. Observing that there is no significant difference between 3WordEx and 3Word (for $\tilde{\lambda}_{10}$, $z = 0.45, p = 0.325$), we combine all variants requiring three words in order to get a larger sample of data. For all the $1,709$ passwords collected from 3WordEx, 3Word, and those in 3or2WordEx that use three words, the observed $\tilde{\lambda}_1$ is 0.2$\%$ and $\tilde{\lambda}_{10}$ is 1.1$\%$, both are significantly less than that from two word variants and the control group (Control), indicating that passwords created using three word variants are likely to be stronger than the baseline.

Four Words. For the $3,307$ total passwords collected from the four words strategy, $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ are 0.06$\%$ and 0.4$\%$. $\tilde{\lambda}_{10}$ observed in the 4 word variant is even smaller than that of 3 word variant ($z = 2.69, p = 0.004$).

Overall, $\tilde{\lambda}_1$ and $\tilde{\lambda}_{10}$ in words selection reduce with the increasing number of words required. Since the entropy introduced in the step converting words into passwords is limited, the results indicate that variants requiring more words provide stronger passwords, where at least three words is sufficient to provide passwords stronger than those chosen by typical users without any strategy specified.

6.4 Study 2: Usability

We conducted a two-phase user study evaluating the usability of the word-based strategies. The study design was almost identical to that described in Chapter 5.4 except that the usability of Control, 3WordEx, and 4WordEx was evaluated in this section.

Table 6.7 and Table 6.8 list the general statistics of the first-phase and second-phase study, respectively. In the first phase, for each condition, we list the number of participants, average password creation time, and statistics for short-term password recall (if applicable) including success rate before and after seeing the strategy as a hint, failure rate after 4 attempts, and time used in password recall. In the second phase, we list the number (percentage) of participants that returned to the study; statistics about long-term password recall, including the number (percentage) of participants who did not write down passwords; the success/failure rate and average time used in password recall for those who did not write passwords down; the number (percentage) of participants who used the strategy provided to update their passwords. In the table, *Succ1* means the the number of participants who successfully recall the password within 2 attempts. *Succ2* means the the number of participants who successfully recall the password in the third or fourth attempts, and the strategy was displayed as a hint. *No WDP* means the number of participants who did not write down passwords. Time is measured in seconds.

6.4.1 First Phase Results

We recruited 425, 388, and 403 participants for Control, 3WordEx, and 4WordEx, accordingly, with a total of 1216 (624 females).

Table 6.7.: Statistics for first-phase usability study for word-based strategies.

Strategy	Short Term Recall	Phase 1					
		Count	Creation Time	Short-term Recall			
				Succ1	Succ2	Failed	Time
Control	Yes	217	38.7	213(98%)	1(0%)	3(1%)	26.0
	No	208	39.8	N/A	N/A	N/A	N/A
	All	425	39.2	N/A	N/A	N/A	N/A
3WordEx	Yes	194	74.9	192(99%)	1(1%)	1(1%)	25.5
	No	194	68.5	N/A	N/A	N/A	N/A
	All	388	71.7	N/A	N/A	N/A	N/A
4WordEx	Yes	202	95.7	200(99%)	0(0%)	2(1%)	28.8
	No	201	92.3	N/A	N/A	N/A	N/A
	All	403	94.0	N/A	N/A	N/A	N/A

Password Creation Time. The average time used in password creation for each strategy is listed in Table 6.7. The password creation time was significantly different among the three strategies. As expected, participants spent the least time when there was no restriction (Control), and time spent in Control is significantly less than that 3WordEx ($t = 10.506, p < 0.001$) and 4WordEx ($t = 12.67, p < 0.001$). The time required for selecting an additional

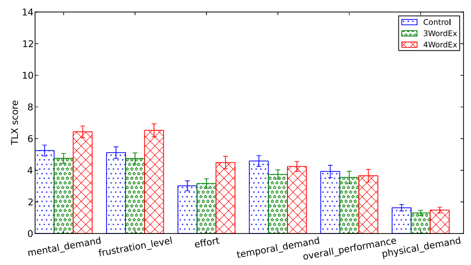
Table 6.8.: Statistics for second-phase usability study for word-based strategies.

Strategy	Short Term Recall	Phase 2						
		Number Returned	Long-term Recall					Update
			No WDP	Succ1	Succ2	Failed	Time	Use Strategy
Control	Yes	169(78%)	140(83%)	51(36%)	9(6%)	80(57%)	43.6	N/A
	No	154(74%)	125(81%)	48(38%)	9(7%)	68(54%)	48.6	N/A
	All	323(76%)	265(82%)	99(37%)	18(7%)	148(56%)	46.0	N/A
3WordEx	Yes	151(78%)	121(80%)	37(31%)	11(9%)	73(60%)	187.8	105(70%)
	No	149(77%)	126(85%)	29(23%)	10(8%)	87(69%)	66.2	107(72%)
	All	300(77%)	247(82%)	66(27%)	21(9%)	160(65%)	125.8	212(71%)
4WordEx	Yes	156(77%)	128(82%)	42(33%)	9(7%)	77(60%)	77.1	102(65%)
	No	155(77%)	116(75%)	25(22%)	5(4%)	86(74%)	75.2	99(64%)
	All	311(77%)	244(78%)	67(27%)	14(6%)	163(67%)	76.2	201(65%)

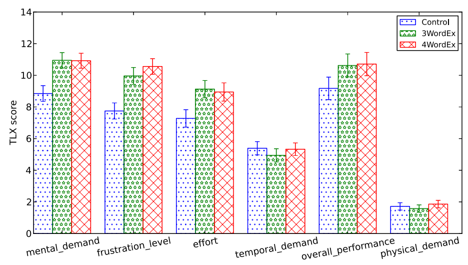
word was significant as well. Time spent in 4WordEx was significantly longer than that in 3WordEx ($t = 4.54, p < 0.001$).

Workload. Fig 6.2(a) shows the average ratings in each subscale of NASA-TLX for the three strategies. Overall, the perceived workload was relatively low, with the average ratings for all subscales being below or close to 7. Among the 6 subscales, physical demand was rated the lowest in general, as it is not critical to the current task. The temporal effort and overall performance among the three strategies were similar, because all participants successfully created their passwords. For the other three subscales, the workload required in Control is similar to that in 3WordEx, indicating that using 3WordEx did not requirement extra workload in creating passwords comparing with the baseline. However, the three subscales for 4WordEx was significantly higher. The difficulty in creating passwords significantly increased when the fourth word was required.

Short-term Memorability. About half of the participants in each strategy were asked to recall the password at the end of the first phase. From Table 6.7, we can observe that regardless of the strategy used, almost all participants could enter the correct password.



(a) Password Generation



(b) Password Recall

Figure 6.2.: Mean scores of TLX as a function of strategy and subscale for Control, 3WordEx, and 4WordEx. Error bars represent standard errors of the scores.

6.4.2 Second Phase Results

Approximately 77% of participants from each strategy condition returned to the second phase of the study.

Long-term Memorability. For participants who came back for the study, approximately 80% indicated that they did not write down the password or the words (in 3WordEx and 4WordEx), and the ratio for the three strategies was similar, indicating that participants were confident that they could remember the passwords without writing them across the strategies, and they did not believe passwords created in word-based strategies were harder to recall. We analyzed the password memorability from the participants who claimed that they did not write down passwords or sentences.

About 37% of participants in Control and about 27% participants using word-based strategies recalled the passwords successfully within first two attempts. When the strategy was displayed as a hint, the ratio of participants were able to recall the passwords increase ranging from 5% to 9% depending on the strategy. Control gave the lowest failure rate (56%) in long-term password recall, followed by 3WordEx (65%), and 4WordEx (67%). The performance of Control was significantly better that from 3WordEx ($\chi^2 = 6.67, p = 0.036$), and 4WordEx ($\chi^2 = 6.54, p = 0.038$), and no significant difference between 3WordEx and 4WordEx was observed ($\chi^2 = 1.42, p = 0.492$). When there was a short-term recall, the long-term recall success rates were generally increased for each strategy, and the increase in rates was larger for the word-based strategies. However, the difference among the strategies was not significant.

Long-term Recall Time. The password recall time in Control was similar to 3WordEx ($t = 1.25, p = 0.21$), but was significantly shorter than that for 4WordEx ($t = 5.25, p < 0.001$). The comparison result is consistent with the comparison of password creation among the strategies. Again, whether or not short-term recall had been required did not have any significant impacts.

Workload. The workload of password retention evaluated by TLX is illustrated in Fig 6.2(b). Comparing Fig 6.2(a) and Fig 6.2(b), perhaps the most noticeable difference is that the subscale of performance in Fig 6.2(b) is almost double that in Fig 6.2(a), which was mainly due to large portion of failed recall. Except physical demand and temporal demand, which are not directly related to the task, the average rates of all the other 3 subscales also increased dramatically, suggesting that password recall was more difficult than password generation.

For the subscales, mental workload and frustration ratings of word-based strategies were higher than those of the Control, suggesting the overhead from the extra requirements of the word-based strategies. No significant difference is observed between 3WordEx and 4WordEx, which is consistent with the comparison in success rate in long-term retention.

Password Update. At the end the task, we asked participants to update the password, without any restriction except that the password could not be the same as the old one. For 3WordEx and 4WordEx, after the password was created, we asked participants whether they used the strategy we provided. About 71% of participants using 3WordEx said "yes" to the question, and the ratio is higher than that for 4WordEx (65%). The results indicated that most of the participants were willing to use the instructed strategy even if not forced to do so, and 3WordEx is more preferable.

Overall, the study suggests that both 3WordEx and 4WordEx perform worse than Control in terms of long-term password retention. Although 3WordEx and 4WordEx have similar failure rate in long-term password retention, paricipants believed 4WordEx was harder to use than 3WordEx. 3WordEx outperform 4WordEx in almost all the measurements. Comparing with 3WordEx, passwords from 4WordEx takes longer time and more workload to create, and participants are less willing to use the strategy.

6.5 Conclusion

In this chapter, we investigated the security and usability of 7 variants of word-based password creation strategies. We showed that popular words, such as "cat" and "dog" were commonly selected among the variants, and the security of the variants depended on the number of words required. Using the standard cracking-based methodology, password sets obtained under all variants were all much more secure than the baseline. However, using β-guess-rates, we found that variants with two words required were likely to produce passwords with similar security level to the baseline. On the other hand, requiring 3 or 4 words significantly reduced the number of word combinations that are chosen more than once, suggesting significantly stronger resulted passwords, where the variant requiring 4 words performs better. In terms of usability, both 3WordEx and 4WordEx performed worse than Control in terms of long-term password retention, and no significant difference between 3WordEx and 4WordEx was observed. However, 4WordEx was considered to be harder to use than 3WordEx. Comparing with 3WordEx, passwords from 4WordEx took longer time and more workload to create, and participants were less willing to use the strategy.

7. SUMMARY

In this dissertation, we improved the eco-system of passwords from multiple aspects.

First, we provided methodology to help password research. We introduced probability threshold graphs for evaluating password datasets. We introduced knowledge and techniques from the rich literature of statistical language modeling into password modeling. We also identified new issues (such as normalization) that arise from modeling passwords, and a broad design space for password models, including both whole-string models and template-based models. Third, we have conducted a systematic study of many password models, and obtained a number of findings. In particular, we showed that the PCFG$_W$ model, which has been assumed to be the state of the art and has been widely used in password research, underperformed whole-string Markov models in our experiments. We expect that the new methodology and knowledge of effectiveness of Markov models can benefit future password research.

Second, we improved the password policies and practice used by websites by addressing the question how to best check weak passwords. In particular, we modeled different password strength checking methods (including password strength meters) as Password Ranking Algorithms (PRAs), and we introduced two metrics: the β-Residual Strength Graph (β-RSG) and the Normalized β-Residual Strength Graph (β-NRSG), to compare them using real world password datasets. In our evaluation, we found unreasonably high frequency of some suspicious passwords. We removed the associated accounts by identifying suspicious account IDs. We then, applied the metrics on cleansed datasets, and showed that dictionary-based PRA had similar performance with the sophisticated PRAs. If the size of PRAs are limited in order to be fit into a client, a hybrid method combining a small dictionary of weak passwords and a Markov Model with backoff with a limited size can provide the most accurate strength measurement.

Finally, we studied the most commonly suggested password creation strategies. We investigate the security and usability of 6 variants of mnemonic sentence-based strategies and 7 variants of word-based strategies. Regarding mnemonic sentence-based strategies, we showed that using the standard cracking-based methodology, password sets obtained under all variants have similar strengths and were all much more secure than the baseline. However, using β-guess-rates, we found that different instructions had tremendous impact on the security level of the resulted passwords. In particular, the instructions for the mnemonic strategy found in the literature and recommended by security experts are not optimal in inducing secure password distributions. However, combining explicit instructions of choosing a personalized sentence that is unlikely to be chosen by others, with the inclusion of corresponding examples, dramatically increased the security of the resulting passwords. For word-based strategies, we showed that popular words, such as "cat" and "dog" were commonly selected among the variants, and the security of the strategies depended on the number of words required. Using β-guess-rates, we found that variants with two words required are likely to produce passwords with similar security level to the baseline. On the other hand, the requirement of 3 or 4 words significantly reduced the number of word combinations that are chosen more than once, suggesting significantly stronger resulted passwords, where the variant requiring 4 words performed better. In terms of usability, both 3-word based strategies and 4-word based strategies performed worse than the control group in terms of long-term password retention. The extra difficulty in password creation introduced by requiring the fourth word was observed.

Overall, in this dissertation, we discussed metrics evaluating password datasets and password models. Based on that, we improved password policies used by websites. If weak passwords are checked and filtered properly, typical users will be encouraged or even forced to create passwords that are less predictable and more crack-resistant. On the other hand, users need knowledge and skills, which is generally known as password creation strategies, to create strong and memorable passwords. We study the security and memorability of passwords induced by several commonly suggested strategies. We expect a combination of proper weak password checking by websites and the skill of creating strong and memorable passwords by users has significant impact on reducing weak and predictable passwords, and therefore improve the eco-system of passwords.

REFERENCES

REFERENCES

[1] C. Herley and P. C. van Oorschot, “A research agenda acknowledging the persistence of passwords,” *IEEE Security & Privacy*, vol. 10, no. 1, pp. 28–36, 2012.

[2] A. Adams and M. A. Sasse, “Users are not the enemy,” *Communications of the ACM*, vol. 42, no. 12, pp. 40–46, 1999.

[3] R. Morris and K. Thompson, “Password security: A case history,” *Communications of the ACM*, vol. 22, no. 11, pp. 594–597, 1979.

[4] D. V. Klein, “Foiling the cracker: A survey of, and improvements to, password security,” in *Proceedings of the 2nd USENIX Security Workshop*, 1990, pp. 5–14.

[5] F. T. Grampp and R. H. Morris, “The unix system: Unix operating system security,” *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 8, pp. 1649–1672, 1984.

[6] D. Florêncio and C. Herley, “A large-scale study of web password habits,” in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 657–666.

[7] S. Riley, “Password security: What users know and what they actually do,” in *Usability News*, ser. 1, B. S. Chaparro, Ed., vol. 8. Software Usability Research Laboratory (SURL) at Wichita State University, 2006.

[8] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez, “Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms,” in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012, pp. 523–537.

[9] 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 *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 2595–2604.

[10] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor, “Encountering stronger password requirements: User attitudes and behaviors,” in *Proceedings of the 6th Symposium on Usable Privacy and Security*, 2010, p. 2.

[11] B. Ur, P. G. Kelley, S. Komanduri, J. Lee, M. Maass, M. Mazurek, T. Passaro, R. Shay, T. Vidas, L. Bauer *et al.*, “How does your password measure up? the effect of strength meters on password creation,” in *Proceedings of the 21st USENIX Security Symposium*, 2012, pp. 65–80.

[12] Y. Zhang, F. Monrose, and M. K. Reiter, “The security of modern password expiration: An algorithmic framework and empirical analysis,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, pp. 176–186.

[13] J. H. Huh, S. Oh, H. Kim, K. Beznosov, A. Mohan, and S. R. Rajagopalan, "Surpass: System-initiated user-replaceable passwords," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015, pp. 170–181.

[14] "John the ripper password cracker," 2014, http://www.openwall.com/john/.

[15] W. E. Burr, D. F. Dodson, and W. T. Polk, *Electronic authentication guideline*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.

[16] M. Weir, S. Aggarwal, M. Collins, and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, pp. 162–175.

[17] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005, pp. 364–372.

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

[19] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from Markov models," in *Proceedings of the Network and Distributed System Security Symposium*, 2012.

[20] C. Castelluccia, A. Chaabane, M. Dürmuth, and D. Perito, "When privacy meets security: Leveraging personal information for password cracking," *arXiv preprint arXiv:1304.6584*, 2013.

[21] X. de Carné de Carnavalet and M. Mannan, "From very weak to very strong: Analyzing password-strength meters," in *Proceedings of the Network and Distributed System Security Symposium*, 2014.

[22] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, "Measuring real-world accuracies and biases in modeling password guessability," in *Proceeding of the 24th USENIX Security Symposium*, 2015, pp. 463–481.

[23] D. Wheeler, "zxcvbn: realistic password strength estimation. Dropbox blog article," 2012.

[24] K. Scarfone and M. Souppaya, "Guide to enterprise password management (draft)," 2009, NIST Special Publication 800-118 (Draft).

[25] "xkcd password generator," 2011, http://preshing.com/20110811/xkcd-password-generator.

[26] B. Schneier, "Choosing secure passwords," 2014, https://www.schneier.com/blog/archives/2014/03/choosing_secure_1.html.

[27] ——, "Passwords are not broken, but how we choose them sure is," 2008, https://www.schneier.com/blog/archives/2014/03/choosing_secure_1.html.

[28] J. Bonneau, “The science of guessing: Analyzing an anonymized corpus of 70 million passwords,” in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012, pp. 538–552.

[29] S. Boztas, “Entropies, guessing, and cryptography,” Department of Mathematics, Royal Melbourne Institute of Technology, Tech. Rep. 6, 1999.

[30] S. Brostoff and M. A. Sasse, ““Ten strikes and you’re out”: Increasing the number of login attempts can improve password usability,” in *Proceedings of the Human-computer Interaction Security Workshop*, 2003.

[31] “How to create a password you can remember,” 2013, http://www.wikihow.com/Create-a-Password-You-Can-Remember.

[32] E. H. Spafford, “Opus: Preventing weak password choices,” *Computers & Security*, vol. 11, no. 3, pp. 273–278, 1992.

[33] F. Bergadano, B. Crispo, and G. Ruffo, “Proactive password checking with decision trees,” in *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997, pp. 67–77.

[34] U. Manber and S. Wu, “An algorithm for approximate membership checking with application to password security,” *Information Processing Letters*, vol. 50, no. 4, pp. 191–197, 1994.

[35] J. J. Yan, “A note on proactive password checking,” in *Proceedings of the 2001 Workshop on New Security Paradigms*, 2001, pp. 127–135.

[36] A. Forget, S. Chiasson, P. C. van Oorschot, and R. Biddle, “Improving text passwords through persuasion,” in *Proceedings of the 4th Symposium on Usable Privacy and Security*, 2008, pp. 1–12.

[37] S. Egelman, A. Sotirakopoulos, I. Muslukhov, K. Beznosov, and C. Herley, “Does my password go up to eleven?: The impact of password meters on password selection,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 2379–2388.

[38] S. Schechter, C. Herley, and M. Mitzenmacher, “Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks,” in *Proceedings of the 5th USENIX Conference on Hot Topics in Security*, 2010, pp. 1–8.

[39] R. Veras, C. Collins, and J. Thorpe, “On the semantic patterns of passwords and their security impact,” in *Proceedings of the Network and Distributed System Security Symposium*, 2014.

[40] M. Dell’Amico, P. Michiardi, and Y. Roudier, “Password strength: An empirical analysis.” in *INFOCOM*, vol. 10, 2010, pp. 983–991.

[41] M. Dell’Amico and M. Filippone, “Monte carlo strength evaluation: Fast and reliable password checking,” in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015, pp. 158–169.

[42] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, “Measuring password guessability for an entire university,” in *Proceedings of the 20th ACM Conference on Computer and Communications Security*, 2013, pp. 173–186.

[43] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse." in *Proceedings of the Network and Distributed System Security Symposium*, vol. 14, 2014, pp. 23–26.

[44] R. Shay, L. Bauer, N. Christin, L. F. Cranor, A. Forget, S. Komanduri, M. L. Mazurek, W. Melicher, S. M. Segreti, and B. Ur, "A spoonful of sugar?: The impact of guidance and feedback on password-creation behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2015, pp. 2903–2912.

[45] S. Komanduri, R. Shay, L. F. Cranor, C. Herley, and S. Schechter, "Telepathwords: Preventing weak passwords by reading users' minds," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 591–606.

[46] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "The memorability and security of passwords: some empirical results," *Technical Report-University Of Cambridge Computer Laboratory*, p. 1, 2000.

[47] J. J. Yan, A. F. Blackwell, R. J. Anderson, and A. Grant, "Password memorability and security: Empirical results." *IEEE Security & Privacy*, vol. 2, no. 5, pp. 25–31, 2004.

[48] K.-P. L. Vu, B.-L. B. Tai, A. Bhargav, E. E. Schultz, and R. W. Proctor, "Promoting memorability and security of passwords through sentence generation," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 48, no. 13, 2004, pp. 1478–1482.

[49] C. Kuo, S. Romanosky, and L. F. Cranor, "Human selection of mnemonic phrase-based passwords," in *Proceedings of the 2nd Symposium on Usable Privacy and Security*, 2006, pp. 67–78.

[50] "Passwords," 2009, http://wiki.skullsecurity.org/Passwords.

[51] S. Chiasson, P. C. van Oorschot, and R. Biddle, "A usability study and critique of two password managers," in *Proceedings of the 15th USENIX Security Symposium*, 2006.

[52] Z. Li, W. He, D. Akhawe, and D. Song, "The emperor's new password manager: Security analysis of web-based password managers," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 465–479.

[53] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, "Password managers: Attacks and defenses," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 449–464.

[54] R. Zhao and C. Yue, "All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*, 2013, pp. 333–340.

[55] ——, "Toward a secure and usable cloud-based password manager for web browsers," *Computers & Security*, vol. 46, pp. 32–47, 2014.

[56] L. Xing, X. Bai, T. Li, X. Wang, K. Chen, X. Liao, S.-M. Hu, and X. Han, "Cracking app isolation on Apple: Unauthorized cross-app resource access on MAC OS X and iOS," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, 2015, pp. 31–43.

[57] W. A. Gale and G. Sampson, “Good-turing frequency estimation without tears,” *Journal of Quantitative Linguistics*, vol. 2, no. 1, pp. 217–237, 1995.

[58] S. M. Katz, “Estimation of probabilities from sparse data for the language model component of a speech recogniser,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, p. 400401, 1987.

[59] J. Bonneau, “Guessing human-chosen secrets,” Ph.D. dissertation, University of Cambridge, 2012.

[60] “CSDN cleartext passwords,” 2011, http://dazzlepod.com/csdn/.

[61] “Openwall wordlists collection,” 2013, http://www.openwall.com/wordlists/.

[62] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, “Passwords and the evolution of imperfect authentication,” *Communications of the ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[63] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.” in *Proceedings of the 2nd ACM Conference on Knowledge Discovery and Data Mining*, vol. 96, no. 34, 1996, pp. 226–231.

[64] M. Burnett, “Today I am releasing ten million passwords,” 2015, https://xato.net/passwords/ten-million-passwords/.

[65] R. Shay, S. Komanduri, A. L. Durity, P. S. Huh, M. L. Mazurek, S. M. Segreti, B. Ur, L. Bauer, N. Christin, and L. F. Cranor, “Can long passwords be secure and usable?” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 2927–2936.

[66] S. G. Hart and L. E. Staveland, “Development of NASA-TLX (task load index): Results of empirical and theoretical research,” *Advances in Psychology*, vol. 52, pp. 139–183, 1988.

[67] 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.

[68] P. S. Foster, K. M. Roosa, V. Drago, K. Branch, G. Finney, and K. M. Heilman, “Recall of word lists is enhanced with increased spreading activation,” *Aging, Neuropsychology, and Cognition*, vol. 20, no. 5, pp. 553–566, 2013.

VITA

VITA

Weining Yang was born and raised in Shanghai, China. He attended Tsinghua University, and graduated with a Bachelor of Engineering in computer science in July of 2011. Weining entered Purdue University in the Fall of 2011, and worked under the supervision of Dr. Ninghui Li in the Department of Computer Science. Weining's graduate work and research was in the area of private data publishing and user authentication. He received his Master of Science in computer science in December of 2013, and his Ph.D. in computer science in August of 2016.