Password Strength Meters: Implementations and Effectiveness

Dalton J. Gusaas
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
gusaa004@morris.umn.edu

ABSTRACT

A password strength meter is a function that takes a user-created password as input and outputs a measure of that password's strength. The general purpose of a password strength meter is to eliminate weak passwords and provide users with feedback and guidelines to help them strengthen their password. This paper will cover three implementations of password strength meters: rule-based meters, an adaptive meter (APSM) [2], and an analyze-modify meter (AMP) [3]. Of these three meters, the AMP is the most effective as it provides an accurate measure of a password's strength, advice on how to further strengthen a password, and modifies its training data every time a user enters a password in order to maintain a uniform distribution of passwords.

Keywords

password strength meters, password checking, password policies, usability, information security, user authentication

1. INTRODUCTION

The usage of passwords is necessary for user authentication and security. Ideally, only randomly generated passwords would be used, as they are the hardest guess [7]. Unfortunately, a randomly generated password is also the least memorable and usable for the majority of users. Since user-created passwords tend to be fairly weak, password strength meters are required to improve the quality of passwords. Since the majority of users do not necessarily know what a strong password is, or simply do not care, it is important for the meter to provide helpful feedback. In order to reduce user frustration and improve the usability of the password, it is also important that the requirements are not too stringent. A user who is frustrated with requirements or does not understand them, will generally choose a weak password that barely meets the requirements [5].

One of the more significant developments for password strength meters was the release of the National Institute of Technology and Standards (NIST) Electronic Authentication Guideline Special Publication 800-63-1; we will refer to this document as SP-800-63-1 [1]. NIST is a government organization that publishes standards and guidelines for government and private usage. Naturally, SP-800-63-1 contained guidelines and recommendations for the development of authentication systems, including a recommendation for measuring password strength based on Shannon entropy [1]. This recommendation served as the basis for many rule-based meters — meters with requirements relating to minimum length, number of uppercase letters, number of digits, and number of symbols [3]. Researchers, including Weir et al., discovered that the NIST recommendation contained significant flaws in estimations of password strength [7]. This gave rise to researchers working on other, better methods of approximating password strength based on probabilistic password cracking methods.

This paper is laid out in four sections. In Section 2 we describe background information required to understand the three discussed password meters as well as a few general terms to help understand password cracking and the measurement of password strength. Section 3 focuses on the description of the three password meters: the NIST SP-800-63-1 document and the resultant rule-based meters, the APSM created by Castelluccia et al., and the AMP created by Houshmand and Aggarwal. In Section 4 the three methods are compared and discussed. In Section 5 we conclude.

2. BACKGROUND

Before moving on to the password strength meters, the necessary background information needs to be introduced. Shannon entropy is described first. Next, the password cracking concepts: guessing entropy, brute-force attacks, and dictionary-based attacks. Finally, the method-specific concepts: n-gram models and PCFGs.

2.1 Shannon Entropy

Shannon entropy can be defined as a way "to determine the randomness of a variable based upon knowledge contained in the rest of the message" [7]. Since Shannon's work dealt with storing information, entropy is measured in bits. The randomness is expressed with the following equation: let x be a random variable and $P(x_i)$ be the probability that x is equal to x_i [3]. The entropy of x, $H(x)$, is then:

$$H(x) = -\sum_{i=0}^{n} P(x_i) log_2 P(x_i) \quad (1)$$

If a character that is present in the language of the message

UMM CSci Senior Seminar Conference, December 2015 Morris, MN.

but, is not present in the "message" it is not included in the summation as the $log_0 = 0$o. For example, given the random string of characters, *mom*: m has a frequency of $\frac{2}{3}$ and o has a frequency of $\frac{1}{3}$. With these frequencies the entropy of a character from *mom* can be calculated with Equation 1: [4]

$$\begin{aligned} H(x) &= -[(0.33 * log_2 0.33) + (0.67 * log_2 0.67)] \\ &= -[(-0.53) + (-0.39)] \\ &= 0.92 \end{aligned}$$

The probabilities of English characters not present in *mom* are not included in this summation as their frequency is 0 and the $log_2(0) = 0$. The entropy of a character chosen at random from m and o for the string *mom* is about 0.92 bits. The total entropy of *mom* is then 3 bits, rounded up from 2.76 bits. Essentially this entropy score measures how random a given string is. Because it applies to strings, Shannon entropy has been applied in cryptography as a way to measure the strength of a password, as the more random a password is the harder it is to guess. There are some issues with using Shannon entropy to measure password strength. In the above example, we assumed that *mom* is a random string of characters. In reality, it is an English word and not a random string of characters. If we switch to this specification, then the entropy becomes 0 bits as there is no other English word that can expressed with these three characters. If a password strength meter assumes the first specification and not the second it would greatly overestimate the strength of *mom* as a password. In order for Shannon entropy to be applicable for passwords, or even English strings, the probability of each character needs to be known. Without knowing the probability and distribution of passwords, the Shannon entropy of a password will not be an accurate measure of the password's strength, as shown in Section 3.1

2.2 Guessing Entropy and Minimum Entropy

Guessing entropy is the approximate amount of work required by an attacker to guess a given password. This measure is often equated with the strength of a password. In practice, the guessing entropy is equal to the number of guesses made on a password before the correct one is guessed. The most efficient strategy for an attacker is then to guess weaker, more common passwords first and stronger, less common passwords later [2].

Minimum entropy is the measurement of the amount of work needed to guess the easiest password in a system. If an attacker is attempting to gain access to a system and not a given user, the minimum entropy is the approximate amount of guesses that need to be made before gaining access to the system as it will be through a high probability password.

2.3 Attack Methods

Though this paper focuses on meters, it is important to briefly discuss a couple of password cracking techniques for comparison of meters and context for why meters developed. A hash is an object created by sending a password through a hash function, a function that attempts to obfuscate what the actual password is. Most password cracking methods work by comparing the actual hash of the password with the hash of a guess until a match is found. Attackers can obtain hashes in multiple ways, such as watching network traffic or breaking into the password hash database. An attacker may also simply enter password guesses into a log-in screen until they are locked out or guess the password.

The first is brute-force attacks. These attacks work by trying every possible combination characters starting with a single character and increasing until it has cracked the password. Though inefficient for long and varied passwords, this method can quickly crack shorter passwords.

The second is dictionary attacks. These attacks use a dictionary of common passwords, or an actual dictionary, to compare the hash of password to entries from its dictionary. Dictionary attacks rely on the assumption that most users choose actual words or phrases as passwords or predictable variations of these words or phrases.

Castelluccia et al. as well as Houshmand and Aggarwal both cited probabilistic cracking methods as the basis for the development of their respective meters. Probabilistic cracking methods combine the best of the above methods. Training on leaked passwords provides a dictionary to base more intelligent brute-force attack that follows the strategy outlined in the previous subsection. The concepts that the probabilistic password strength meters discussed in this paper rely on are n-grams and PCFGs.

2.4 n-gram Models

An n-gram is a sequence of n consecutive items taken from a larger sequence [8]. For example, the sequence of English characters *password* can be broken down into a number of English character n-gram sequences. The 1-gram sequence of *password* would be: p, a, s, s, w, o, r, d and the 2-gram sequence would be *pa*, *as*, *ss*, *sw*, *wo*, *or*, *rd*.

An n-gram model is a sequence of n-grams that can be used to determine the probability of the next occurring n-gram based on the probability distribution, i.e. the frequency of each n-gram, gathered from training the model on sequences in a language [8]. For a given sequence of n-grams $x_{i-(n-1)}, ..., x_{i-1}$, the n-gram x_i can be predicted. This can also be written as:

$$P(x_i | x_{i-(n-1)}, ..., x_{i-1}) \tag{2}$$

Lets say we have a language made up of the 1-grams a, e, l, t, and r. Furthermore lets say that these 1-grams can combined into 3-grams that must be English words, e.g. *eat*, *let*, and *ate*. Using all the different combinations of the 1-grams into 3-grams a probability distribution can be determined to guess what a given 1-gram is based off of the preceding 1-gram. For example, given the a what is the most probable 1-gram to follow? In this case only three of the five 1-grams (l, t, r) can be used to form actual three-letter words. From these three 1-grams the words *ale*, *ate*, *art*, and *are* can be formed. This gives us a probability distribution for the 1-gram following a to be $l = 0.25$, $t = 0.25$, and $r = 0.50$. The most likely 1-gram to follow a would be r. As shown in Section 3.2, n-gram models can be a powerful for guessing passwords, especially if they are trained on actual passwords.

2.5 Probabilistic Context-Free Grammars

A context-free grammar is a 4-tuple (V, Σ, R, S) where:

1. V is a finite set called the variables,
2. Σ is a finite set, disjoint from V, called the terminals,
3. R is a finite set of rules, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable. [6]

For example, lets say we have a grammar G with starting variable A, variables B and C, and terminals x, y, and z. The grammar is formalized as G = ({A, B, C}, {x, y, z}, {set of 3 rules}, S = A).

$$A \Rightarrow xBz$$
$$B \Rightarrow A|xCz$$
$$C \Rightarrow y$$

From these substitution rules derivations can be made, such as xyz which looks like:

$$A \Rightarrow xBz \Rightarrow xxCzz \Rightarrow xxyzz$$

String xBz is substituted for starting variable A, string xCz is substituted for variable B, and finally terminal y is substituted for string xCz.

CFGs are a powerful tool for modeling languages, adding probabilities allows for the modeling of languages such as passwords. A probabilistic CFG is the same as a CFG except that its definition has an added set – making it a 5-tuple instead of a 4-tuple. The added set, P, is the set of probabilities on substitution rules [10]. By assigning probabilities to the rules in the previous example CFG, it becomes a PCFG. Adding a 100% probability that A substitutes for B ($B \Rightarrow A$) means that only derivations $x^n z^n$, such as xxxzzz, occur. Section 3.3 shows that applying PCFGs to passwords allows for effective prediction of password layouts by determining the probabilities of certain patterns.

3. METHODS

In this section, we go over three different implementations of password meters. The first, rule-based meters, are the most widely used with a presence in the sign-up pages for Google, Facebook, and the University of Minnesota, Morris. The other two have not been encountered due to only recently been proposed by researchers. Note that the studies and reports cited here generally assume that an attacker is not targeting a single user, but is either trying to gain access to any user in the system or gain access to the system itself [2, 3, 7]. Therefore, the meters are used in order to strengthen a system's security as well as the user's.

3.1 Rule-Based Meters

3.1.1 Description

The NIST Electronic Authentication Guideline SP-800-63-1 [1], which rule-based meters are based off, reasoned that the Shannon entropy of a password can be used as a starting point to approximate the entropy of a given password. The authors noted that this is a difficult assumption to make given the jump from English words, which Shannon used in his work, to passwords [1]. The approximation is broken down into a set of rules developed by transferring Shannon's assumptions on a 27 character alphabet, a-z and the space, to the 94 character alphabet that is present on standard QWERTY keyboards. The following rules are taken directly from SP-800-63-1: [1]

1. The entropy of the first character is taken to be 4 bits;
2. The entropy of the next 7 characters are 2 bits per character; this is roughly consistent with Shannon's estimate 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 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 require both upper case and non-alphabetic characters.
6. A bonus of up to 6 bits of entropy is added for an extensive dictionary check.

NIST intended these rules to provide a rough estimate of a password's entropy. The idea was the rules would be used in combination with other security measures so the rough estimation of password strength would not be an issue.

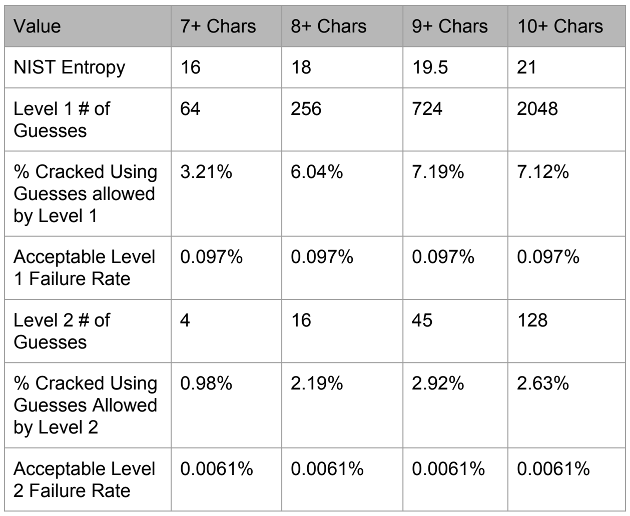
With these rules, the password *Daltong!u* would have a total entropy of 25.5 bits; 4 bits for *D*, 14 bits for *altong!*, 1.5 bits for *u*, and a bonus 6 bits for having uppercase and non-alphabetic characters *D* and *!* — if the password had only one of these it would not get the bonus. This score does not factor in a dictionary test which could increase the score by up to 6 bits. From these six rules, rule-based meters were derived, often including rules such as a minimum length of eight, at least one uppercase letter, at least one symbol, and at least one digit — all of which have precursors in the NIST rules. These rules act as a guide or requirement to help users create passwords that are more difficult to guess. Rule-based meters serve as a reasonable method for improving password strength; however, they were knowingly built on a rough estimation of password strength, arguably making them unsuitable for high security environments.

3.1.2 Weaknesses

The greatest weakness of the NIST guideline is that it did not have access to a large corpus of user-created passwords to base its assumptions on [7]. As a result, several of its assumptions and guidelines do not reflect reality. In their study of NIST entropy, Weir et al. [7] used the leaked RockYou password database in combination with the password cracking software John the Ripper[1] to test if the NIST assumptions held up with real passwords. RockYou is a company that creates games for various social media websites such as Facebook and Myspace [9]. In 2009 an attacker was able to gain access to the RockYou password database and leaked 32 million passwords. The RockYou password leak is perfect for this study as it provided a large corpus passwords. Furthermore, since RockYou was integrated with numerous websites, the leaked passwords do not share a uniform creation policy. This means the RockYou leak contains a better representation of user behavior than a password leak from a uniform password creation policy.

One assumption disproved was that the Shannon Entropy of a password could be used to estimate its guessing entropy. The writers of SP-800-63-1 made it clear that this assumption was a rough one and should only be used a rule of thumb. Nonetheless, SP-800-63-1 defined two levels of acceptable risk. For Level 1 the chance of an attacker randomly guessing a password should be less than 0.097% and

[1]Official website: http://www.openwall.com/john/



Value	7+ Chars	8+ Chars	9+ Chars	10+ Chars
NIST Entropy	16	18	19.5	21
Level 1 # of Guesses	64	256	724	2048
% Cracked Using Guesses allowed by Level 1	3.21%	6.04%	7.19%	7.12%
Acceptable Level 1 Failure Rate	0.097%	0.097%	0.097%	0.097%
Level 2 # of Guesses	4	16	45	128
% Cracked Using Guesses Allowed by Level 2	0.98%	2.19%	2.92%	2.63%
Acceptable Level 2 Failure Rate	0.0061%	0.0061%	0.0061%	0.0061%

Figure 1: Testing the NIST rules against a simulated attack using password cracking software John the Ripper. Taken from [7].

for Level 2 the chance should be less than 0.0061% [7]. These levels were supposed to allow systems to tailor their security based on the following equation, where $H(x)$ is the entropy of a given password:

$$\text{Level 1: Number of Allowed Guesses} = 2^{H(x)} * 2^{-10}$$

$$\text{Level 2: Number of Allowed Guesses} = 2^{H(x)} * 2^{-14}$$

Thus, even with a rough entropy measure it would be possible for defenders to ensure the security of their system by limiting the number of allowed guesses for a given password based off its entropy. As Figure 1 shows, the percentage of cracked passwords is well above the acceptable amount. The number of guesses in Figure 1 correlates to the upper limit of guesses before an attacker would be locked out of an individual account. This disproves the assumption of Shannon entropy equaling, or even being a rough estimate, of guessing entropy. For this assumption to have held the percentage of cracked passwords should have been below the acceptable level of failure.

Other assumptions that Weir et al. investigated related to the distribution and composition of actual passwords. For example, instead of each symbol or digit having an equal probability, certain characters such as *!* and *123* were shown to be much more prevalent then others. Furthermore, it was discovered that certain structures are much more common. If a password had a special character in it, 28.20% of the time there would be only one special character and it would be at the end of the password. Though these are only a couple of examples, the study ultimately concludes that users tend to make predictable passwords that follow patterns. This means that passwords are not uniformly distributed across the standard available key-space, allowing attackers to tailor an attack if they know what the rules are for password creation. The results also demonstrate that the rules set out by NIST were a naive application of Shannon entropy, as they had no actual password data to back up the entropy scores provided by their six rules. Without knowing the actual distribution of passwords or the probability of each character the NIST specification is unable to accurately measure the strength of passwords.

3.2 Adaptive Password Strength Meters

Building off the weaknesses of NIST SP-800-63-1, as outlined by Weir et al., Castelluccia et al. proposed a new password meter called an Adaptive Password Strength Meter (APSM) [2]. The APSM works by using n-gram models to predict the next character that will occur given a string of length m. Thus a string can be broken down into its n-grams. The probability of a given string $c_i, ..., c_m$ can then be written as the product of the probability of the ith character given the preceding characters.

$$P(c_i, ..., c_m) = \prod_{i=0}^{m} P(c_i | c_{i-n+1}, ..., c_{i-1}) \tag{3}$$

For example, the probability of string *hello* can be broken down into its sub-probabilities.

$$P(hello) = P(h)P(e|h)P(l|he)P(l|hel)P(o|hell)$$

The meter is also tied to a database that keeps track of the n-gram counts, allowing the APSM to base its prediction of the next character on the collected n-grams. Whenever a new password is added, for each present n-gram in the password, the *count* of the n-gram in the database is incremented. Using *are* as a password would increment the n-grams of *a*, *r*, *e*, *ar*, *re*, and *are* by one. Additional noise is added to the n-grams database in the form of additional n-grams unrelated to the entered password. This noise has only a small impact on meter performance and increases the overall security of the actual passwords should there be some form of data breach. This is important because if an attacker were able to get access to an n-gram database that did not add noise, they would be able to tailor an attack using the present n-grams. Equation 3 can be rewritten as a fraction with the numerator as the count of the next probable n-gram and the denominator as the count of the current n-gram. The *count()* function returns the count of the given n-gram.

$$\begin{aligned} &P(c_i | c_{i-n+1,...,c_{i-1}}) \\ &= \frac{count(c_{i-n+1}, ..., c_i)}{count(c_{i-n+1}, ..., c_{i-1})} \\ &= \frac{count(c_{i-n+1}, ..., c_i)}{\sum_{x \epsilon \Sigma} count(c_{i-n+1}, ..., c_{i-1}, x)} \end{aligned}$$

The previous equation can be further simplified so that *f(c)* equals the approximate strength of a given password c

$$f(c) = -log_2(\prod_{i=0}^{m} P(c_i | c_{i-n+1}, ..., c_{i-1})) \tag{4}$$

Castelluccia et al. used the 5-gram from *password*, *asswo*, as an example to show that the probability of an *o* following *assw* in the RockYou database is 0.97.

$$p(o|assw) = \frac{count(asswo)}{count(assw)} = \frac{98450}{101485} = 0.97$$

The overall probability of *password* was then calculated to be 0.0016%. The actual occurrence of *password* in the RockYou database was 0.0018%. With such a close approximation of frequency, these results suggest the meter can accurately recognize weak passwords. Since this meter bases its measure of strength off of passwords in the system, it can be used to

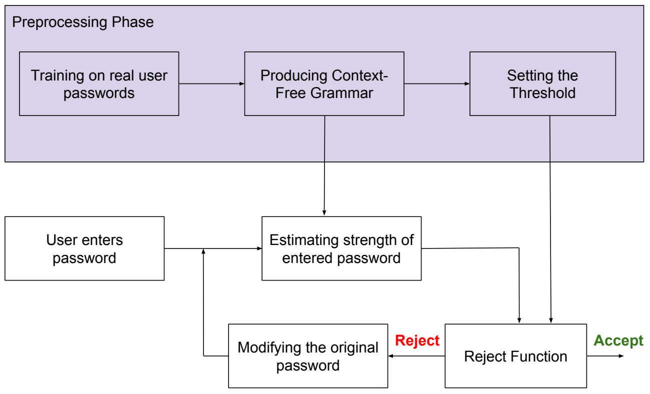

Preprocessing Phase
Training on real user passwords
Producing Context-Free Grammar
Setting the Threshold
User enters password
Estimating strength of entered password
Modifying the original password
Reject
Reject Function
Accept

Figure 2: Flow chart of the AMP. Taken from [3].

maintain uniform distribution of passwords, making it more resistant to attacks. [2]

3.3 Analyzer and Modifier for Passwords

The Analyzer and Modifier for Passwords (AMP) is a of password meter created by Houshmand and Aggarwal [3] that uses PCFGs to measure the strength of passwords. The PCFG is composed of the components: L for alphabet letters, D for digits, S for symbols (!, @, %, etc), and M for uppercase letters. In this PCFG, the password *Password123!* is represented as a group of components $M_1L_7D_3S_1$; a collection of these components is also known as a base structure. To make this a true PCFG, the probabilities are computed by determining the frequency of the base structures in the training data. The training data is taken from the numerous password leaks that have occurred in the last decade, including the RockYou leak. The probability of alphabet letter components, e.g. L_7, is not determined by the training data as the probability would not be considered a sufficient sample size. Instead, the probability of a given L_n is $\frac{1}{x}$ where n is the length of the string and x is the number of strings of that length that appear in a dictionary. For base structures and variables that do not occur in the training data, Houshmand and Aggarwal used probability smoothing to assign low probability values so the PCFG still accounts for them. As shown in Figure 2, the training and generating of the PCFG are the first steps in the preprocessing phase.

Next the system calculates a threshold value, T, such that passwords with a higher probability than T are weak and those with lower probability are strong. This is the last part of the preprocessing phase in Figure 2. T is equal to the probability used by an optimal attack, i.e., guessing high probability passwords in decreasing order of probability. Since the PCFG provides an accurate measure of password distribution, the number of guesses needed to reach T, $G(T)$, can be determined. The researchers calculated $G(T)$ in two ways. The first creates a table of guesses paired with probabilities and time intervals. Since the guesses are actually being computed, this way is rather slow, despite being accurate. The second provides a conservative lower bound for $G(T)$ until T is reached. Since it uses the generated PCFG to find the lower bound of $G(T)$ it is much faster and nearly as accurate as the other approach.

Starting with "User enters password" in Figure 2, the flow of the system can be followed. Unlike other meters, the AMP does not flat out reject a password, it instead modifies a rejected password using its distance function. The goal of the function is to generate a modified password that is over the threshold value and is one edit distance, a single change, different from the original. The researchers reasoned that an edit distance of one would maintain the usability and memorability of the password, and in most cases be stronger than the threshold. The researchers defined two sets of operations for modifying weak passwords divided between operations on base structures and operations on components. For base structures the operations are insertion, deletion, and transposition (swapping two adjacent components). For components the operations are insertion, deletion, substitution, and case (inverting the case of one letter in a component). Before the operations can be applied the rejected password is broken down into its base structures and put at the root node of a tree structure. From the root node, child nodes are randomly chosen until a leaf node is reached. Each leaf node is the result of any number of operations performed on the original password. If the change made at this node puts the password above T then the system is done and the user is handed that suggested password. The random movement in the tree may seem inefficient, but it is meant to diversify the passwords suggested by the system.

Now that the major parts of the AMP have been outlined, let's walk through the system step by step as a user. None of the probabilities or edits in this example reflect the exact operation of the AMP, instead they are meant to demonstrate how it works. We start by entering the password, *Dalton123!*. The password is then parsed into the base structure $M_1L_5D_3S_1$. The probability of the password is equal to the product of the probabilities of the base structures and components. For this password it is:

$$P(Dalton123!) = P(M_1L_5D_3S_1)P(M_1)P(L_5)P(D_3)P(S_1)$$

For this example let's say that *Dalton123!* has a high probability in comparison with the threshold value and is rejected. The password is then passed to the distance function which puts it at the root node of a tree and moves randomly until it ends on the leaf node with **123***Dal***T***o***N***!*, see Figure 3 for an example tree. This resulted from one base structure transposition and two component case changes, totaling an edit distance of three. This edit distance would be unusual for the AMP as often only a single change needs to be made to strengthen a password.

Houshmand and Aggarwal noted that after a period of usage it is possible that the PCFG may obtain a non-uniform password distribution, invalidating the threshold value and making the system vulnerable to an attack. They addressed this by dynamically updating the grammar with each new password, which keeps the grammar uniform and helps ensure that the modifier suggests less probable passwords. The actual training data is not changed in this process, instead the probabilities of each base structure is updated. [3]

4. DISCUSSION

In comparison to the rule-based meters, the probabilistic meters described in this paper are more accurate at assessing password strength. One benefit that rule-based meters do have is the relative ease of implementation as the simplest rule-based meters do not require training data or a database to store probabilities. Rule-based meters need to simply check a password against a set of required rules. For

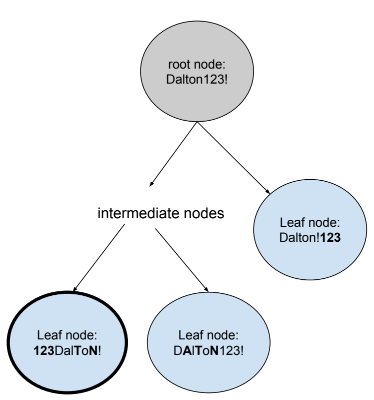


Figure 3: Example of how a password could be changed by the AMP distance function.

websites and accounts that do not require a high-level of security rule-based meters are fine. Even with lockout conditions, such as locking an account after three incorrect log-in attempts or blocking log-in attempts from a specific IP, rule-based meters do not provide the same level of security as probabilistic meters. By enforcing a set of rules that all passwords in the system must adhere to, system administrators provide attackers with information to optimize their attacks. If passwords need to follow a set of rules, the number of potential passwords is limited, which allows attackers to only guess passwords that would be allowed by the password strength meter. Since probabilistic strength meters aim to uniformly distribute passwords across the allowed key space, they do not suffer from this issue.

Of the two probabilistic meters discussed in this paper, it is difficult to discern which provides a higher level of security. The AMP certainly provides a better experience for users in suggesting similar alternatives for weak passwords, though this functionality could probably be added to the APSM since they function in a similar manner. Both meters estimate strength using training data that is modified as new passwords are analyzed in addition to being based on probabilistic models. These two meters also are resistant to the attack methods described in Section 2.3. A simple brute force attack will have difficulty with the distribution of characters, a dictionary attack may fair slightly better but will not be able to optimize its training data, and probabilistic attacks will have no advantage as long as the probabilistic meters maintain a uniform distribution of passwords.

5. CONCLUSION

The probabilistic password meters proposed by Castelluccia et al. as well Houshmand and Aggarwal look to be promising improvements upon the rule-based meters set down by the NIST SP-800-63-1. Systems implementing probabilistic meters are more resistant to attacks as it is more difficult for the attacker to tailor their method to a certain distribution of passwords as the passwords created in the presence of a probabilistic meter will have a uniform distribution. Even if the n-gram database of the training PCFG were to be leaked, it would not give the attacker any useful information. For users of a system with probabilistic meters, there is no decrease in the usability of their passwords. As shown in Section 3.3, the AMP has little impact on user-created passwords, often able to strengthen a password by suggesting only a single change. Furthermore, since all strength measurements are based on a threshold value, the meter can be tuned to be more or less stringent depending on the required security of the system. Of the three password meters discussed in this paper, in terms of effectiveness, flexibility, and usability the AMP as laid out by Houshmand and Aggarwal appears to be best implementation of a password strength meter.

Acknowledgments

Thanks to Kristin Lamberty, Nic McPhee, and Brian Goslinga for providing feedback and helping me through this process.

6. REFERENCES

[1] W. E. Burr, D. F. Dodson, E. M. Newton, R. M. Perlner, W. T. Polk, S. Gupta, and E. A. Nabbus. Nist Special Publication 800-63-1 Electronic Authentication Guideline. Technical report, National Institute of Technology and Standards, 2006.

[2] C. Castelluccia, M. Duermuth, and D. Perito. Adaptive Password-Strength Meters from Markov Models. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, NDSS '12, San Diego, CA, USA, 2012. NDSS.

[3] S. Houshmand and S. Aggarwal. Building Better Passwords Using Probabilistic Techniques. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 109–118, New York, NY, USA, 2012. ACM.

[4] L. Kozlowski. Shannon entropy calculator, 2015. [Online; accessed 19-November-2015].

[5] 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 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 173–186, New York, NY, USA, 2013. ACM.

[6] M. Sipser. *Introduction to the Theory of Computation – Third Edition*. Cengage Learning, Boston, MA, 2013.

[7] 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*, CCS '10, pages 162–175, New York, NY, USA, 2010. ACM.

[8] Wikipedia. N-gram — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 25-October-2015].

[9] Wikipedia. Rockyou — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 19-November-2015].

[10] Wikipedia. Stochastic context-free grammar — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 25-October-2015].