Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords

Matt Weir, Sudhir Aggarwal, Michael Collins, Henry Stern
Florida State University, Redjack LLC, and Cisco IronPort Systems
weir@cs.fsu.edu, sudhir@cs.fsu.edu, michael.collins@redjack.com, hestern@cisco.com

ABSTRACT
In this paper we attempt to determine the effectiveness of using entropy, as defined in NIST SP800-63, as a measurement of the security provided by various password creation policies. This is accomplished by modeling the success rate of current password cracking techniques against real user passwords. These data sets were collected from several different websites, the largest one containing over 32 million passwords. This focus on actual attack methodologies and real user passwords quite possibly makes this one of the largest studies on password security to date. In addition we examine what these results mean for standard password creation policies, such as minimum password length, and character set requirements.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection – *Authentication*

General Terms
Security, Human Factors, Measurement

Keywords
Password Cracking, Cybercrime, Password Policies

1. Introduction
Secure password generation is complicated by the tradeoff between developing passwords which are both challenging to crack and usable. Truly random passwords are difficult for users to memorize, and user-chosen passwords may be highly predictable. Password policies attempt to mediate between these two goals by forcing users to incorporate additional complexity into a password, such as by mandating the user include an odd character or use passwords of some minimal length. However, these policy mechanisms are hampered by an ill-defined understanding of their actual effectiveness against real attack techniques, and by circumvention strategies employed by the users. For example, a policy mandating that a user include at least three digits in a password will often result in the user simply appending "123" on the end of an insecure password. A sufficiently sophisticated password cracker will be aware of these strategies and incorporate mechanisms for breaking them into their tools.

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

It is unlikely any other document has been as influential in shaping password creation and use policies as the NIST Electronic Authentication Guideline SP-800-63 [1]. The findings and recommendations published in it have proven the basis for many government and private industry password policies [2, 3]. Central to this document is the notion of measuring password entropy. The idea of information entropy was first formalized by Claude Shannon [4] as an approach to measure the amount of information that is unknown due to random variables. In a way, it attempts to determine the randomness of a variable based upon knowledge contained in the rest of the message. Most often this randomness or information is expressed using the following equation:

$$\mathbf{H(x)} = -\sum_{i=0}^{n} \mathbf{P(x_i)}\,\mathbf{Log_2}\,\mathbf{P(x_i)} \qquad \mathbf{1}$$

For example, a fair coin flip would land as heads 50% of the time. The resulting entropy of modeling a single fair coin flips would then be $-\sum_{i=0}^{1} \mathbf{P(.5)}\,\mathbf{Log_2}\,\mathbf{P(.5)}$ which is equal to 1 bit of entropy. Each successive flip of the coin would add an additional bit of entropy as the result is a summation across all of the variables x_i. The context for Shannon's research was to determine the amount of lossless compression that can be performed to store or transmit a message. In the previous case, the smallest message on average that could be sent describing the results of a run of completely fair coin flips would require at least one bit of data per coin toss.

What the NIST standard did was attempt to use the concept of Shannon's entropy for estimating the strength of password creation policies against online password cracking attacks. As we'll show in the following sections, this unfortunately is not a valid approach. While the Shannon entropy value would be useful to determine on average the minimum amount of space required to store or transmit a human generated password, it has no relation to the *guessing entropy* of a password. To put it another way, even with an accurate Shannon entropy value, it would not tell the defender anything about how vulnerable a system would be to an online password cracking attack.

If the Shannon entropy value is not useful when determining the strength of a password creation policy, then the question remains, what is the benefit of different password creation rules? Is an eight character password on average stronger than a seven character password, and if so, by how much? Even in the NIST SP800-63 publication when attempting to gauge how much entropy is added to a system due to various creation policies, the authors themselves state, "*Unfortunately, we do not have much data on the passwords users choose under particular rules.*" A related paper authored by a Microsoft researcher [5], which attempted to gauge the security provided by strong passwords, raised much the same point: "*As far as we are aware, there is no data available on strength related attacks on passwords of web-sites that maintain lockout policies."*

Our approach then in this paper is twofold. First, we demonstrate that the use of Shannon's entropy as defined in NIST SP800-63 is not an effective metric for password security. Second we attempt to gauge the security provided by conventional password creation rules. We accomplish both of these tasks by performing standard password cracking attacks against multiple sets of real life passwords. These passwords, which will be described in more detail in Section 4, and Appendix 1 and 2, were all obtained from publicly disclosed hacking attacks. This is where an attacker collected the passwords, either through a phishing attack, or compromising a website, and for whatever reason posted the password lists online. These lists in some cases can be quite large, as in the RockYou set [6] which contained over 32 million passwords. Admittedly these datasets can be problematic, since none of them represent corporate logins. A counter-example can easily be made that people on average choose stronger passwords for more sensitive sites. That being said, these datasets still represent a significant number of user password creation strategies and can be applied to evaluate the expected success rate of different types of attacks. We hope this focus on real passwords and real attack methodologies can provide a better understanding of the effectiveness of different password creation policies.

The remainder of this paper is structured as follows: Section 2 details some of the previous work done in this area. Section 3 covers the NIST SP800-63 model of password entropy. Section 4 illustrates why that the NIST notion of password entropy does not provide an accurate view of password security. Section 4 further goes on to demonstrate the effectiveness of password cracking strategies against traditional password creation rules. Finally Section 5 discusses password creation policies that might be more applicable when defending against online attacks.

2. Previous Work

There have been several previous attempts to measure password security by analyzing real life passwords. One of the first papers to take this approach was written in 1978 by R. Morris and K. Thompson [19]. They found that in a group of around 3,000 users, 1/3rd of the passwords were vulnerable to a dictionary attack containing 250,000 words. When combined with a limited brute force attack, they estimated over 86% of the passwords could be cracked. Since then several other studies have found similar results. In [20], A. Narayanan, and V. Shmatikov ran experiments against 142 real user passwords and were able to break 67.6% of them using a Markov based brute force attack. In [21], Yan, Blackwell, and Anderson found when testing a group of 300 student passwords, 32% of the control group was cracked via a limited dictionary based attack. In [22], Wu collected over 25 thousand Kerberos v4 tickets and attempted to crack the corresponding user passwords. In that experiment, only 8.1% of the passwords were cracked over a two week period due to the computational complexity of making a password guess. Perhaps the largest previous study on password security was done by Stone-Gross et al when his team temporarily took over the torpig botnet [23]. During the ten day period they had control of the botnet, their group collected over 297 thousand unique username/password pairs from 52 thousand infected computers. To test the strength of the plaintext passwords collected, they hashed 173 thousand unique passwords with the MD5 hashing algorithm and then proceeded to use the popular password cracking tool John the Ripper to try and crack the hashes using an offline attack. During the course of a 75 minute cracking session, the team managed to break over 40% of the passwords. What's more, they found that 28% of users re-used the same password across multiple sites. This closely matches an earlier study by Sophos [24], where 33% of users polled admitted to using the same password for all of their internet logins. If this holds true, that means passwords gathered from low value targets, such as social networking websites, might successfully be used by an attacker to target higher value targets such as webmail and bank accounts. It also means that the results of studying these "low value" passwords may provide us insight into the effectiveness of password creation policies for higher value sites.

That being said, none of the above studies focused specifically on the security that password creation policies actually provide, such as the effect password length has on password strength. There has been some research into how effective the notion of Shannon entropy is for measuring password strength, (and by extension the recommendations put forward by NIST 800-63). The most notable papers covering the subject have been [7, 8], but those studies focused exclusively on the theoretical underpinnings of trying to convert the Shannon entropy to the Guessing entropy of a system, and did not verify their findings using real user passwords. In the pessimistically titled paper, "Password Exhaustion: Predicting the End of Password Usefulness" [25], Clair et al, attempted to evaluate the search spaces produced by different password creation policies along with their resistance to attack. They found that certain password policies might actually weaken systems against brute force attacks due to the reduction in key space. They then collected 3,500 student passwords and attempted to crack them using a 20 node cluster of computers. This resulted in their team breaking 34% of the passwords in five days, with a vast majority of these passwords, (almost 90% of the cracked passwords), falling to brute force attacks. Unfortunately, their tests did not attempt to measure security provided, (or reduced), by the application of different password creation policies beyond their resistance to brute force attacks. Therefore, we feel that the results and strategies detailed in this paper are fairly novel as we attempt to gauge the security of password creation policies by examining real user passwords and their resistance to dictionary based attacks.

One other paper that bears mentioning is a survey of password creation and storage policies among several popular websites by J. Bonneau and S. Preibusch [26]. There are too many interesting findings from that paper to list here, and it is highly recommended reading to help put the results detailed later in this paper into context with how password policies are currently implemented. For example, a vast majority of the websites Bonneau and Preibusch examined, including sites such as eBay, Amazon.com, and Wordpress, did not support rate limiting the number of guesses allowed to an attacker.

3. Password Entropy per NIST SP800-63

As mentioned previously, the password recommendations provided in the NIST document are based on the idea of information entropy. Building on the notion of entropy detailed in Equation #1, it can further be expanded by noting that the entropy of several random variables can be modeled as:

$$\mathbf{H}(\mathbf{x}, \mathbf{y}) \leq \mathbf{H}(\mathbf{x}) + \mathbf{H}(\mathbf{y}) \qquad \mathbf{2}$$

In the NIST document, they attempt to define these random variables by specifying how they are created through the use of common password creation policies. These random variables can be viewed as representing an unknown value that an attacker

would have to guess when attempting to crack a password. Each variable is assigned an entropy score, and the sum of all the entropy scores is added up to create a final entropy total for the entire system using Equation 2. The entropy score for each variable is assigned using the following criteria which is quoted directly from the original NIST paper:

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 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 password, 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.

There has been some confusion based on the examples given in the NIST document whether rule #5 requires numbers and special characters to both be present or if the presence of either one would allow assigning of the "bonus" six bits of entropy. If such a distinction is important in any of the tests in this paper, the method to calculate rule #5 will be explicitly stated.

As an illustration of using the above model , consider a password creation policy requiring nine character passwords, and for at least one uppercase letter, lowercase letter, digit, and special character to be present. The resulting NIST entropy score would then be calculated as 25.5 bits of entropy as seen in Fig 3.1.

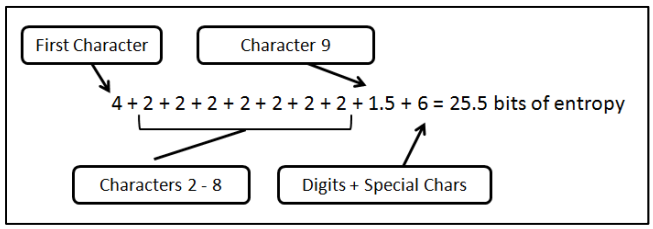

Figure 3.1: Example Calculation of NIST Entropy

The next step is to translate the resulting entropy score into a measure of security provided by the password creation policy. If the entropy calculation is equivalent to a uniformly distributed randomly generated key of length H(x), this becomes possible due to the following equation:

$$\text{Chance of success} = \text{Number of Allowed Guesses} / 2^{H(x)} \quad (3)$$

For example, an attacker will have a 50% chance of cracking a random key if they can search half of the key-space. If the security threshold of the risk allowed is known, (aka it is acceptable for the key to be successfully guessed with a certain probability), Equation 3 can easily be rewritten to then provide the maximum number of guesses that an attacker should be allowed while maintaining a set level of risk. The NIST SP800-63 document does this by designating two different levels of acceptable risk. For a password policy to meet a Level 1 certification, the chance of an attacker succeeding should be 1 in 1024. For a Level 2 certification, the attacker's success rate should be limited to 1 in 16,384. The resulting number of guesses allowed an attacker would then be:

$$\text{Level 1: Number of Allowed Guesses} = 2^{H(x)} \times 2^{-10} \quad (4)$$

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

Given these equations, and the ability to calculate H(x), the idea is to allow a defender to tailor their password creation policy to the level of security that is required and the expected number of guesses that an attacker would be allowed, (either via password lockout policy, or rate limiting). The potential problems with this approach are 1) Does the calculation for H(x) correspond to user behavior, and 2) Is the *Shannon Entropy* value equivalent to a uniformly distributed random key for security purposes? Question #2 was addressed in [7, 8], and it has been shown that the Shannon Entropy value is not equivalent to guessing entropy value. The reason behind this is that the distribution of values is not uniform across the key-space. A quick counter-example to demonstrate this: For Equation 4 to hold true when applied to passwords, the proportion of passwords cracked would have to increase linearly with the number of guesses made. The examples shown later on in Section 4 clearly show this is not the case.

In [7], an alternate metric for measuring password strength was proposed, but as the authors admit, it is only applicable for situations such as one-time keys, and not for human generated passwords intended for day-to-day use. Therefore because the Shannon Entropy does not correlate to the Guessing Entropy, this paper will focus on answering question #1, and more specifically, how much security is provided by existing password creation rules, such as specifying a minimum password length, or requiring a digit. In addition, we will also refer back to the SP800-63 notion of entropy to further provide empirical evidence of its shortcomings to back up the proofs already laid out in [7, 8].

4. Effectiveness of Entropy against Password Cracking Attacks

In this section, we evaluate the entropy scoring provided by NIST SP800-63 document against common password cracking techniques and real life datasets. The intent of this analysis is to provide additional insights into the effectiveness of password creation policies, and to evaluate the value of the SP800-63 notion of entropy in regards to actual password cracking attacks.

4.1 Experiment Methodology

To evaluate the security provided by password creation policies we need to be able to model the threats against them. For this

paper we are primarily concerned about online password cracking attacks, where the security of the system is still operational. In an online attack, the attacker selects a set of *targets* and applies a number of *guesses* in an attempt to crack them. Each guess is applied sequentially against each target, and a target is removed from evaluation when a guess matches the target's password. In the context of this analysis, a "guess" is therefore a guess against each target in the target set. That is, if we state that we are applying "500 guesses", we mean 500 guesses per target.

This model allows us to score password attacks as a function of success per guesses invested. We expect that most online systems follow a per-target lockout policy, meaning that if more than some threshold of attacks are made against a target in a limited period, that target *and only that target* are temporarily locked out. The cost of making guesses against two targets is therefore no more than the cost of making a guess against a single target.

We can slightly modify our approach if the defender uses a global lockout policy (*i.e.,* if the defender locks out *all* targets if attacks exceed a certain threshold). If all of the targets are of equal worth, an attacker will spend their time making the most probable guesses against each target. Therefore the attack strategies remain the same. The one difference is our simulation of per-target lockout policy presents a slightly optimistic view for the defender if a global lockout policy is in place since in a real life attack, if an attacker manages to crack a password they will stop making guesses against it and use their time to focus on the remaining passwords instead. That caution also applies to the rest of the results in this paper, as it is always possible for an attacker to come up with attack methods that performs better than the ones we modeled.

The majority of tests in this paper use passwords collected from the RockYou password list [6]. The RockYou set was originally obtained by an attacker who utilized a SQL injection attack against the rockyou.com website, and then later posted the passwords online. RockYou provided applications for numerous social networking sites such as Facebook, MySpace, and Friendster, and thus included the associated login details created by users for those sites. The actual list itself contained over 32 million passwords. Due to the list's size, we used the GNU shuf tool to randomize it and then divided the list into 32 sub-lists containing one million passwords each. This allows us to run experiments against a smaller set of passwords, while leaving other sub-lists untainted for use in future tests. Using standard machine learning techniques, currently we have assigned the first five sub-lists, RockYou_test1 through RockYou_test5, as part of a test set, while the last five sub-lists, Rockyou-train28 through RockYou_train32 for use in training new attacks. Therefore we are not training and testing against the same passwords.

Because the RockYou list represents passwords collected from a diverse set of web sites, there is no single password creation policy that affects *all* of the words in the list. We can therefore examine the resulting distribution of these passwords as evaluated by the SP800-63 entropy score. To model this, the maximum entropy value for a creation policy which a password could be generated under was evaluated using the rules NIST put forward, with Rule 5 being interpreted as requiring numbers, special characters and uppercase. In addition, it is assumed that no blacklists of forbidden passwords were used. The resulting graph measuring the maximum entropy score for passwords from the RockYou_test1-3 sets can be seen in Fig 4.1.1.

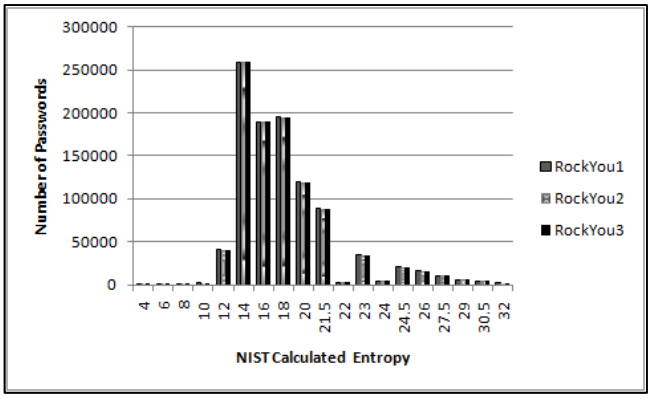

Figure 4.1.1: Maximum SP800-63 Password Creation Policies of the RockYou Data-Set

The NIST entropy values displayed ranged from 4, (the minimum possible under the SP800-63 rules), to 32. The highest entropy value detected was 278, (which may have represented a password created by a spammer or a junk value that was somehow imported into the database). The above results also give a general breakdown of the password length distribution, since very few passwords managed to meet Rule #5. This is also why there were very few passwords which had an entropy value of 22 or 24. Since there was no uniform password creation policy in effect, the results in Fig 4.1.1 are likely to be a reasonable estimate of "normal" user password generation.

4.2 Minimum Password Length, Digits, and Why Password Entropy is not Valid

The next step is to attempt to evaluate the impact of individual password creation policies on the success of password cracking attacks. The first creation policy we will examine is minimum password length. Since we did not have examples of users creating passwords under different length password creation policies, we attempted to model these policies by further dividing the test lists based on the minimum length passwords found in them. Therefore a "7+ sub-list" would contain all the passwords from the parent list that were seven or more characters long.

For the first test, we wanted to show a longer password cracking session that would probably only be performed in an offline attack. While the SP800-63 document is more concerned with online attacks, we felt that the results were useful for comparison, and for modeling other threats. To simulate an offline attack, the popular cracking tool John the Ripper [9] was used, along with its "Single mode" rule-set, (a collection of rules which create a much longer password cracking session than their default rule-set). Since this is a dictionary based attack, the input dictionary used was Dic-0294, which is a commonly used password cracking dictionary [10]. This dictionary contained 869,228 unique words, and the number of guesses was significantly higher due to every mangling rule being applied to each applicable word. Since it can be assumed that the attacker has knowledge of the password creation policy, we did not permit John the Ripper to make guesses shorter than what was allowed for the list it was targeting. The results of running these cracking sessions against the RockYou_test1 list divided in subsets by minimum password length can be seen in Fig 4.2.1. The percentage cracked refers to the number of passwords cracked in each individual sub-list, and not of the total set.

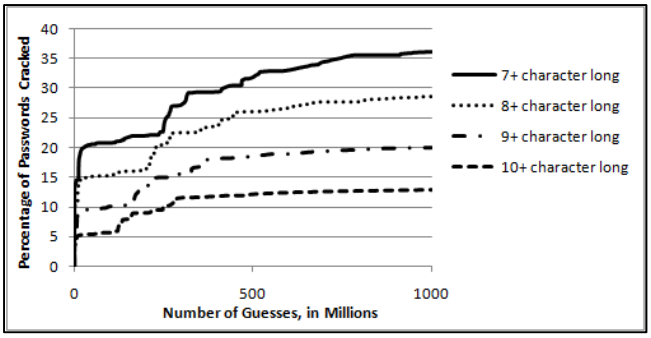

Figure 4.2.1: Results of Minimum Password Length on Longer Password Cracking Sessions

Fig 4.2.1 shows that the effectiveness of each attack was influenced by the minimum password length; however before we examine the impact of minimum password length we must consider several other possible causes that may have influenced this decrease in password cracking effectiveness. One possible issue is that the division of the lists based on minimum password length was done artificially, since the RockYou site did not enforce a uniform password creation policy. This might lead to an unequal distribution of the security minded users between the different sub-lists based on password size. For example, people who created longer passwords when they didn't need to, (based on policy), might inherently choose a stronger password than users who create the simplest password allowed. Since none of the test sets we have access to enforced a password policy requiring users to create passwords longer than six characters long, verifying if this occurs is problematic. Therefore the best we can do is evaluate the composition of passwords in the different sub-lists to see if there are any noticeable differences in password creation strategies between the different sub-lists beyond minimum password length The results of this can be seen in Table 4.2.1.

Table 4.2.1 Password Information from the RockYou1 List

Character Set	7+ Chars	8+ Chars	9+ Chars	10+ Chars
Contains Digits	57.5%	59.5%	60.2%	60.0%
Contains Special Characters	4.4%	5.1%	6.6%	8.0%
Contains Uppercase	6.5%	6.7%	6.9%	7.1%
Contains Only Lowercase Letters, Digits	89.2%	88.4%	86.7%	85.1%

As expected, as shorter passwords were excluded, the average complexity of the remaining sets increased. Of note is that as the length of the passwords increased, their composition also becomes more complex. From 7+ character passwords to 10+ character passwords, the probability of a special character nearly doubles, and the probability of uppercase characters increases by 10%. These results imply that users picking longer passwords were also either forced by policy or preference to add additional security features. To further confirm this, the above lists were once again divided to only include passwords containing at least one digit. Likewise, the password cracking attacks were modified so they only produced guesses that contained digits as well. This test has an additional bonus of allowing us to evaluate the effectiveness of requiring a digit in a password creation policy since we can compare it to the previous test in Fig 4.2.1. The results of re-running the password cracking attacks when digits are required can be seen in Fig. 4.2.2.

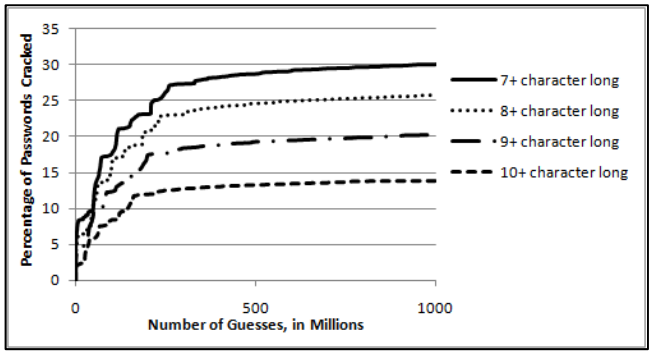

Figure 4.2.2: Results of Minimum Password Length and Requiring Digits on Longer Password Cracking Sessions

The test shows, that even when all passwords sets were required to contain digits, there still was a noticeable difference in the effectiveness of the attacks against the different test sets divided by minimum password length. That being said, the divergence between the test sets was less than if no digits were required; The attack against the 7+ character set performed worse when digits were required, and the attack against the 10+ character set surprisingly performed slightly better when digits were required. Note: it would probably be the wrong conclusion to assume that requiring digits would make 10 character long passwords weaker. What more likely happened was that this additional rule excluded "junk" passwords in the list that did not correspond to an actual user generated password, (for example some very long URLs were present in the RockYou list). In addition several passwords may have been created in a way that made them strong but did not include digits, such as certain pass-phrases. By excluding digits, some of these strong passwords may have been excluded as well. The biggest difference though, from a defender's perspective, is that the requirement of a digit to be present significantly decreased the effectiveness of the password cracking attack in the first hundred million guesses.

Looking at how users incorporated digits into their passwords, Table 4.2.2 shows the most popular digits used in the RockYou_Training32 list.

Table 4.2.2: Top Ten Digits Found in the RockYou32 List

Rank	Digit	Percentage
#1	1	10.98%
#2	2	2.79%
#3	123	2.29%
#4	4	2.10%
#5	3	2.02%

Rank	Digit	Percentage
#6	123456	1.74%
#7	12	1.49%
#8	7	1.20%
#9	13	1.07%
#10	5	1.04%

This means that the top 10 digits counted for roughly 26% of the total digits used. If we further mandate that a password must include an alpha character as well, (a-zA-Z), the results in Table 4.2.2 remain mostly the same, with the main changes being that the number '123456' drops out of the top 10, and that the number '1' by itself becomes more frequent with it appearing 15.44% of the time. Likewise, the coverage of the top 10 digits increases to 36.25% of the total count.

Of course, what is also important is how those digits are actually used in the password itself. For example the digit could be appended at the end of the password, 'secret123', before the

password, '123secret', or used in the middle of the password, 's3cr3t'. To better understand how people incorporate digits into their passwords, Table 4.2.3 shows where the digits appeared in seven character or longer passwords taken from the RockYou_Train32 password list.

Table 4.2.3: How Digits are used in 7+ Character Passwords

Location	Example	Percentage
All Digits	1234567	20.51%
After	password123	64.28%
Before	123password	5.95%
Other	passw0rd, pass123word, p1a2ssword, …	9.24%

It shouldn't come as a surprise that most people simply add digits to the end of a base word when creating a password. If we filter the training list to only include passwords that also contained one non-digit, the number of users who appended a digit to the end of their password jumped to 77.46%. When running additional tests, we found that an attacker could crack 11.06% of 7+ character passwords which included at least one digit and one non-digit, just by appending the number '1' to the end of their password guesses. In an attempt to gain a better understanding of how people used digits that fell into the 'other' category, we performed a manual analysis of a number of the passwords in the training set. We found a vast majority of them used digits as letter or word replacements. Word replacements such as replacing 'for' with '4', and 'to/too/two' with '2' seemed particularly popular. Several other strategies, such as using keyboard combinations, for example '1qaz2wsx', and ASCII art, 'alice<3bob', were also noticed. There were also several passwords where a digit was followed by a special character, such as 'password1!'. While most replacements, (except for the most commonly used ones such as 'passw0rd'), are unlikely to be targeted in an online attack, an attacker can certainly make use of this knowledge when conducting an offline dictionary attack.

The next step is to evaluate what effect increasing the minimum password length has on the effectiveness of an online password cracking attack. To accomplish this, we must first determine a relevant method to compare the success of different password cracking attacks to each other. This is not a trivial problem. One approach would be to compare the final number of passwords cracked. A problem with this approach is: how long should the cracking session be limited to? As an example of that, Fig 4.2.3 shows the same cracking sessions detailed in Fig. 4.2.1, but this time limited to 50 thousand guesses.

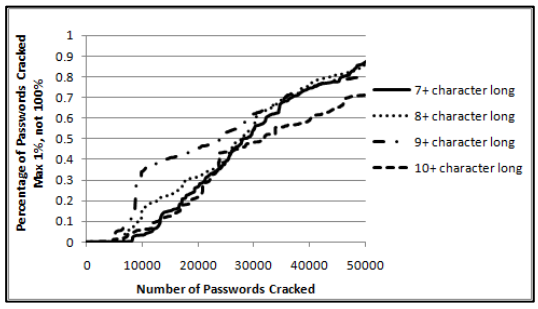

Figure 4.2.3: Results of Minimum Password Length on Shorter Password Cracking Sessions, Un-optimized Attack

At first glance, the results depicted in Fig 4.2.3 are decidedly mixed with the minimum length only making a noticeable difference in the effort required to crack 10+ character long passwords. This is deceiving though, since the attack was not intended/optimized for that short of a password cracking session. As evidence of that, the input dictionary itself, dic-0294, contained over 800 thousand words. This shows the importance of measuring attacks that are tailored to the number of guesses allowed. It is interesting though that even in this un-optimized attack, each of the cracking sessions nearly cracked 1% of the total passwords. We'll see in the next test though that, 1% cracked over 50,000 guesses is actually a poor success rate and we can do much better using an optimized attack.

Another method to measure the relative effectiveness of a password cracking session would be to determine how many guesses were required to crack a certain percentage of passwords. This approach can also be problematic, as it is highly dependent on the percentage set. This can be seen in the previous figures 4.2.1 and 4.2.2, as a password cracking session will start out fast, but quickly slow down as it takes more and more guesses to crack each successive password.

To that end, since the SP800-63 document is primarily concerned with defending against online password cracking attacks of a limited number of guesses, it is useful to see the success rate of a password cracking attack specifically tailored for shorter attack runs. To set the next test up, we combined the training password lists RockYou_train28 through RockYou_train32 to create a training list of five million user passwords. This entire training list was then sorted by the number of occurrences of each password, (so the new list's first password was '123456'), with all duplicate guesses then being removed. This resulted in an input dictionary where the most probable guesses are tried first. Using this targeted input dictionary, we once again ran attacks against the RockYou_test1 list. The results of these attacks when limited to 50k guesses using this targeted dictionary can be seen in Fig 4.2.4. The reason a limit of 50k guesses was selected was that it seemed a reasonable number to model an online attack against a system that did not possess a lockout policy. Therefore if an attacker could make a guess against each account every 10 seconds, this would represent a cracking session of around six days.

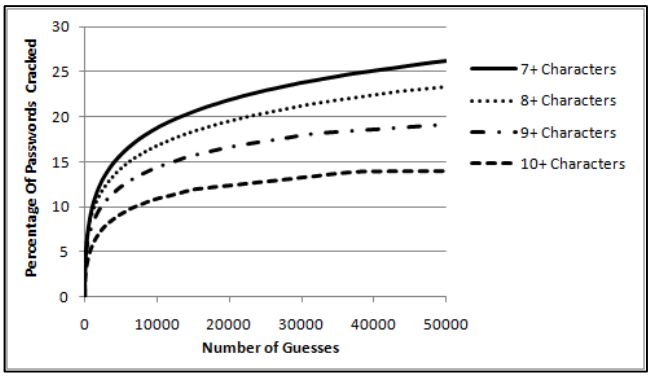

Figure 4.2.4: Results of Minimum Password Length on Shorter Password Cracking Sessions, Optimized Attack

What's striking about the above graph is given just fifty thousand guesses; over 25% of the 7+ character long passwords were cracked. Even the 10+ character password suffered a crack rate of over 14%. Regardless, the session depicted in Fig. 4.2.4 represents a much longer cracking session than what the NIST 800-63 model would recommend. For a Level 1 certified system with a

minimum password length of 10 characters, the maximum number of guesses recommended to allow an attacker and still maintain an acceptable failure rate is 2048 guesses. To better compare the results to the NIST model, Fig 4.2.5 shows the same cracking session, but this time limited to 2,000 guesses.

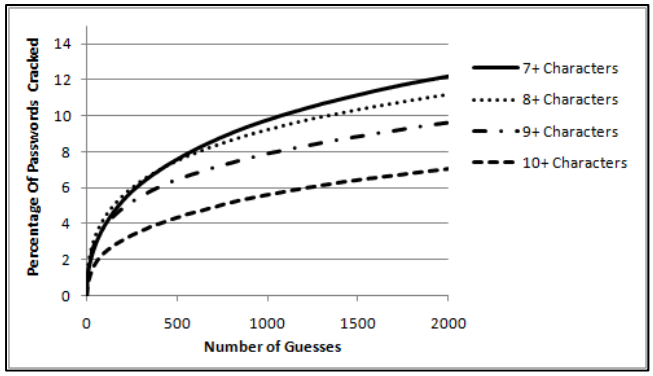

Figure 4.2.5: Results of Minimum Password Length on a Very Short Password Cracking Session

This raises the question, how does the above cracking sessions compare to the acceptable NIST failure rates given for level 1 and 2 certified systems? The answer, based on the above tests can be seen in Table 4.2.4. As the results show, if a blacklist of common passwords is not part of the password creation policy, the NIST model quickly breaks down. For comparison, a policy that meets a Level 1 system should only allow an attacker to guess a password within the allowed number of guesses with a probability of 1 in 1024, or approximately a 0.097% chance. Likewise, a Level 2 certified system should only allow an attacker to be successful with a probability of 1 in 16,384, or approximately a 0.0061% chance. This is drastically different from our findings. What's worse, the attacker's success rate actually increases as the number of guesses allowed grows due to the higher minimum password length, (though it does drop a bit when 10+ characters are required). This implies that the current NIST measurements overestimate the security provided by increasing the minimum password length.

Table 4.2.4: Targeted Cracking Attack vs. the NIST Entropy

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 Level1 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 Level2 Failure Rate	0.0061%	0.0061%	0.0061%	0.0061%

What this shows is that a blacklisting approach must be used for any sort of security against online password cracking attacks, considering that nearly 1% of 7+ character long passwords were cracked with just four guesses. In all fairness, the NIST 800-63 document does state that a blacklisting approach was required for a minimum entropy estimate. That slightly contradicts their original entropy calculation though, where they assign an optional 6 bits of guessing entropy if a blacklist is employed. As we'll see later, even with a blacklisting approach, the NIST model of entropy simply does not hold. In fact, no amount of adding or modifying values when calculating the NIST Entropy can allow it to be effectively applied to model the success rate of a password cracking session against human generated passwords. To illustrate this, Fig 4.2.6 shows the previous test in Fig 4.2.4, but this time also depicting the expected cracking success rate, (using Equation #5), per the NIST entropy model for policies requiring at least seven character long passwords. Equation #5 simply is a rewriting of Equation #3 in a way that can be easily graphed, and takes into account that a seven character minimum length password creation policy would have a NIST entropy score of 16.

$$\text{Chance\_of\_success}(y) = \text{Number\_of\_guesses}(x)/2^{16} \qquad 5$$

Fig 4.2.6 shows conclusively that the NIST notion of entropy does not hold when modeling a password cracking session. For example, the expected success rate given to Equation #5 means 100% of the passwords should be cracked given just 65,536 guesses. This is plainly not the case for any sort of real life attack. What's dangerous about the NIST entropy measurement is that it overestimates the security of certain passwords that may be cracked quickly by the attacker, leaving the defender with a false sense of security, while drastically underestimating the security of many passwords that for all intensive purposes are resistant to an online attack. This in turn can lead to overly burdensome password creation policies that disallow many passwords that in practice would be secure against a determined attacker. In short, the entropy value doesn't tell the defender any useful information about how secure their password creation policy is. While you can change some values in the NIST calculation around, for example saying that each additional character gives 1 bit of entropy vs. 2, there is simply no way to transform a Shannon entropy value to take advantage of Equation #4 and provide the expected success rate of a password cracking attack against human generated passwords.

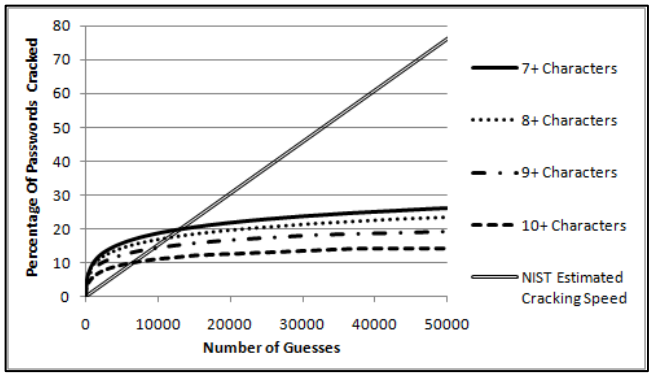

Figure 4.2.6: Comparing the NIST Estimated Cracking Speed vs. a Real Life Attack.

4.3 The Effectiveness of Using Blacklists of Banned Passwords

The next question is how effective are different sized blacklists of prohibited passwords. This was simulated by creating blacklists based on the training list input dictionary from the previous test. For example, a blacklist containing 500 banned passwords would be formed from the 500 most frequently used passwords in the training set. The results of re-running the attacks depicted in Fig. 4.2.4 against the RockYou_test1 7+ character long list, and using various length blacklists, can be seen in Fig 4.3.1. For this cracking session, we also allowed the attacker to have knowledge of the blacklist so they will not waste their time making guesses included in it. In addition, a limit of 50k guesses was once again used to simulate an online cracking session if there was no lockout policy.

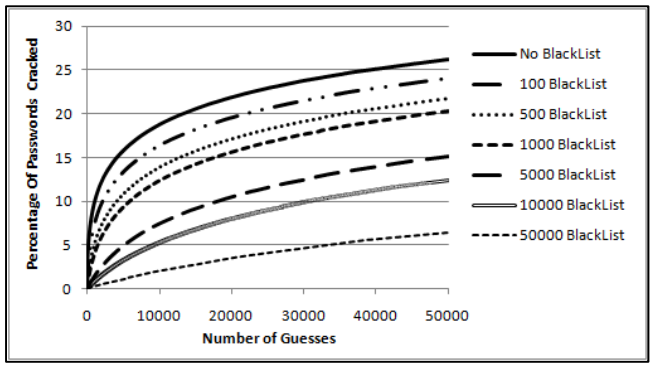

Figure 4.3.1: Results of Using Blacklists When Attacking 7+ Character Long Passwords of the RockYou1 Test List

The most restrictive password creation policy in this test, where passwords must be seven characters long and a blacklist of 50,000 words is used, yields a NIST calculated entropy value of 22. For a Level 1 certified system, that means an attacker would be able to make 4096 guesses, and for a Level 2 system the attacker would be able to make 256 guesses. To better illustrate this, Fig. 4.3.2 shows the above password cracking session limited to 4,000 guesses.

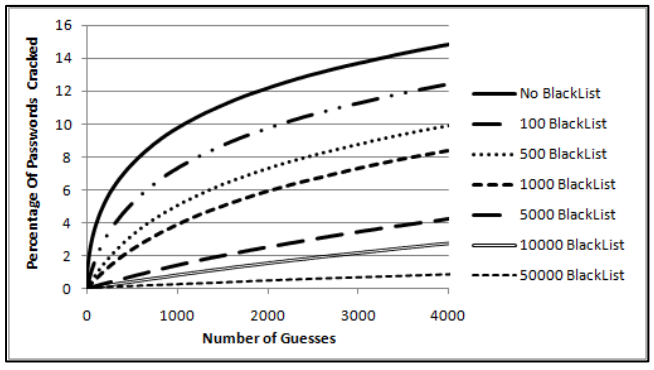

Figure 4.3.2: Results of Using Blacklists When Attacking 7+ Character Long Passwords – Shorter Cracking Session

As the results in Fig 4.3.2 show, given a blacklist of 50k banned words, and the 4096 guesses allowed by a level 1 certified system, an attacker would still be able to crack 0.848% of the passwords, which is still much higher than the 0.097% failure rate that was predicted by the NIST model. For a level 2 certified system, an attacker would be able to make 256 guesses, and in the above test would crack 0.058% of the passwords. This is also significantly higher than the NIST predicted 0.0061% failure rate. On the positive side, while the results show further flaws in NIST approach of calculating password entropy, these tests also demonstrate that a significant improvement in the security of a system may be obtained with even a moderately sized blacklist.

4.4 The Effectiveness of Requiring Uppercase Characters

The next password creation policy we are going cover is case mangling. Fig. 4.4.1 shows a cracking session against a subset of the RockYou_test1 list of all the passwords that contained at least one uppercase character, with the attacker using the RockYou training input dictionary. The attacker once again has knowledge of this password creation policy, and makes use of this when generating their guesses.

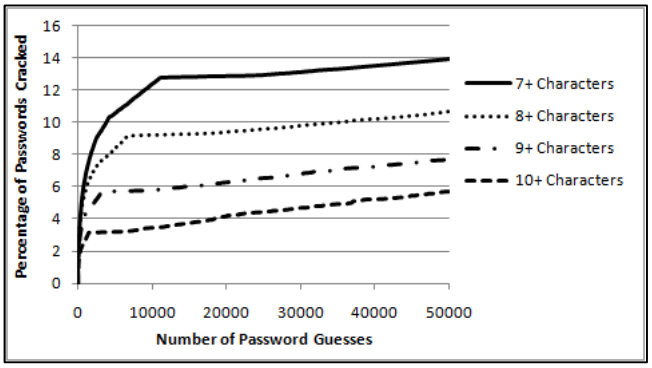

Figure 4.4.1: Attack when Uppercase is required vs. RockYou_test1

What immediately sticks out is that the password cracking sessions start out much like the other attacks, but quickly hit a plateau where they become significantly less effective. Unfortunately this means that there still are a sizable number of users who pick weak passwords and would be compromised in an online cracking attack. On the plus side, this seems to imply that even a moderately sized blacklist would provide significant protection. Looking into the possible causes for this plateau, Table 4.4.1 shows the top ten case mangling rules for seven character long alpha strings that were extracted from passwords with at least one uppercase character. These alpha strings were extracted from the RockYou_training28-32 sets and are independent of the rest of the password (aka Table 4.4.1 only displays the letters, not the digits or special characters that may have also been present in the password). As Table 4.4.1 shows, an attacker could target around 89% of the seven character long alpha strings by either trying all uppercase letters or just capitalizing the first character. This is not inconsequential since in an online attack that would double the number of guesses an attacker would have to make, but it is hardly ideal. This also means that if an attacker has no additional knowledge of the target, they most likely would never try any of the other mangling rules in an online attack. That information still doesn't fully explain the results in Fig 4.4.1 though. As a side note, the reason a string of all lowercase characters appears in the above table is because some of the passwords had an uppercase character separated by another value, such as "P#assword".

Table 4.4.1: Top Ten Case Mangling Rules for 7 characters

String: U=Upper, L=Lower	Probability
UUUUUUU	53.56%
ULLLLLL	35.69%
ULLLULL	1.05%
LLLLLLL	1.03%
ULLLLLU	0.90%
ULLULLL	0.85%
ULULULU	0.68%
LLLLLLU	0.62%
UULLLLL	0.61%
UUULLLL	0.59%

To see if people who used uppercase characters created stronger passwords in other ways vs. people who didn't use uppercase characters, the RockYou_training28-32 passwords were divided by if they contained capital letters or not. The results of a few simple metrics comparing the two lists can be seen in Table 4.4.2.

Table 4.4.2: Comparison of Lowercase vs. Uppercase

Metric	No Uppercase Characters	Only Passwords that Contained an Uppercase
Average Length	7.86 characters	8.28 characters
% that Contained a Digit	53.93	55.74
% that Contained a Special Char	3.15	7.87

This shows there is a small but noticeable increase in the use of other mangling rules when a password is capitalized. That doesn't mean that everyone who used uppercase letters picked a strong password. For example in the RockYou training set input dictionary, the top five passwords that contained uppercase characters were, "PASSWORD", "ILOVEYOU", "PRINCESS", "ABC123", and "Princess". This simply implies that the plateau may in part be due to a larger proportion of users applying other low probability mangling rules or base words.

4.5 The Effect of Requiring Special Characters

Now we move on to password creation policies that require special characters, (aka not uppercase, lowercase or digits). The results of running cracking sessions against the RockYou_test1 set, where only passwords containing a special character were counted, is shown in Figure 4.5.1.

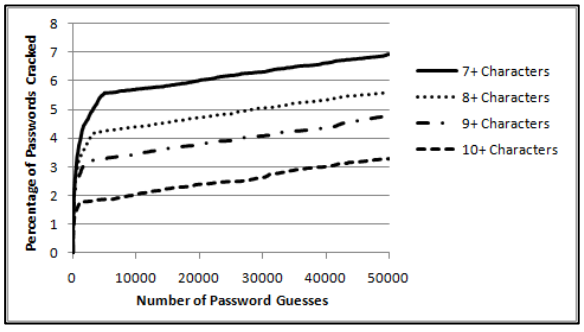

Figure 4.5.1: Attack when a Special Character is Required

Just like with case mangling, the attack starts out fast, but also quickly plateaus. It should be noted that when special characters were required, even in the initial first thousand guesses the attack was much less successful then with the previous password creation rules, such as requiring a digit or capitalized character. Beyond the previously mentioned reasons for why these passwords may be more resilient to attacks, (such as the users who chose special characters creating stronger passwords in general), the variation at which people choose special characters almost certainly has something to do with their strength as well.

Table 4.5.1 shows the top 10 single letter, (aka not part of a keyboard combo such as "!@#$%"), special characters used. This data was collected from the RockYou_training28-32 lists.

Table 4.5.1: Top Ten One Letter Special Characters

Special Character	Probability
.	17.81%
_	14.72%
!	11.34%
-	10.25%
<space>	8.72%
@	7.19%
*	6.54%
#	3.92%
/	3.01%
&	1.84%

This is in stark contrast with digits, where for example 35% of the time if a single digit was selected, that digit was the number '1'. The next step is to see how special characters were used in the password itself. Table 4.5.2 shows the top 10 structures of seven character passwords from the RockYou_Train32 password list that included at least one special character.

Table 4.5.2: Top Ten Structures for Special Characters

String: A=Alpha, D=Digit, S=Special	Probability
AAAAAAS	28.50%
AAASAAA	7.87%
AAAASDD	6.32%
AAAAASD	6.18%
AASAAAA	3.43%
AAAASAA	2.76%
AAAAASA	2.64%
SAAAAAS	2.50%
ASAAAAA	2.38%
AAAAASS	2.17%

As expected, the most common structure simply had a special character appended to the end of it. What's interesting is that it was much more frequent for a special character to be followed by a digit than the other way around. This may be an indirect result of password change policies, where people became used to incrementing their base password by one each time they are forced to select a new password. Looking at the individual

passwords which contained a special character in the middle of them, underscores, dashes, and periods were very popular to break up two words. For example: "ash_lee". Letter replacements, such as replacing an 'a' with an '@', were much less common.

4.6 Comparing the Attacks against Other Password Lists

Now a legitimate concern with the previous tests is that while the training passwords and the test passwords were different, they both came from same website. It can be argued that this represents an unfair advantage for the attacker, and would not translate to a real world password cracking session. To that end, using the same dictionary formed from the RockYou training set, password cracking sessions were run against several other disclosed password lists using a maximum of 50k guesses. For more detail about these lists please see Appendix 1 and 2. As in the previous tests, these lists were divided by minimum password length. The results of these attacks can be seen in Fig. 4.6.1.

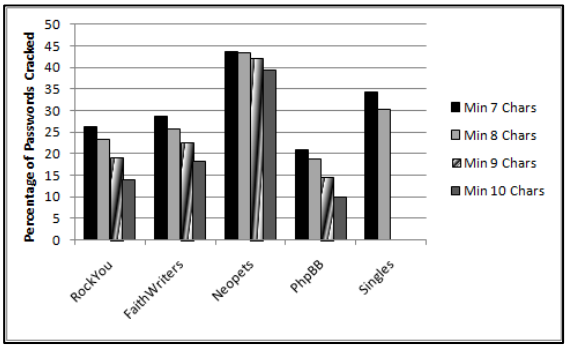

Figure 4.6.1: 50k Guesses against Various Password Lists

As can be seen, the cracking sessions against the RockYou_test1 dataset actually performed worse than most of the other password lists. Much of this is probably due to user training, and the relative importance of the passwords to the users. For example, the Neopets list for the most part represents young children. On the other hand, the PhpBB list was a development and distributions site for the PhpBB bulletin board so most of its users were webmasters and/or programmers.

Another point of interest is the effectiveness of the RockYou training set dictionary compared with other available input dictionaries. To illustrate this, several cracking sessions were run against the FaithWriters password list. Since the FaithWriters and the Singles.org password lists were both composed of people almost exclusively of the Christian faith, another targeted input dictionary was created from the Singles.org list. This dictionary was generated in the same way as the dictionary from the RockYou training list. Since Singles.org represented a much smaller password list, it only contained 12,234 unique words. Likewise the default dictionary provided with John the Ripper, passwords.lst was also used. Due to password.lst's very small size, (3,116 words), the default John the Ripper mangling rules were also applied to generate some of the extra guesses required. All three input dictionaries were then allowed to make 10 thousand password guesses against the FaithWriters list. The results of this can be seen in Fig. 4.6.2.

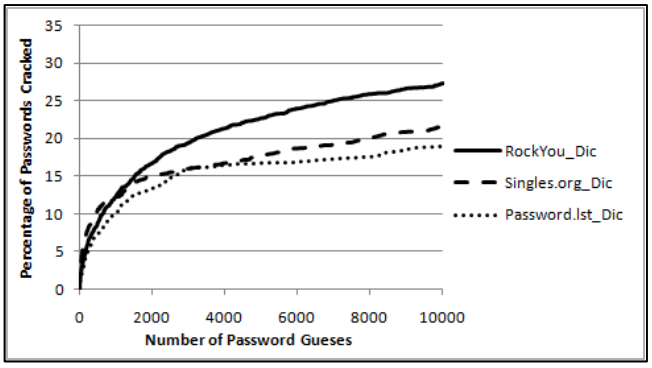

Figure 4.6.2: Different Dictionaries vs. the FaithWriters List

As expected, the Singles.org dictionary performed slightly better initially, but it was too small of a sample size to compete with the custom RockYou dictionary over a longer password cracking session. Both custom dictionaries performed better than the default John the Ripper dictionary. This test illustrates that the findings depicted in Section 4 can always be improved on as an attacker gains more knowledge about their target. It also shows the severity of when a password list the size of the RockYou list is disclosed in how it can improve online password cracking attacks against other sites.

Now it can certainly be argued, (with good reason), that the results found in the previous tests would not hold true if an attacker was targeting corporate or bank passwords. One counter-example to that criticism would be to point out that the popular micro-blogging website Twitter was once compromised due to a site administrator choosing the password 'happiness' [27]. Needless to say, administrators and corporate employees are still human. That is just one example though, and we freely admit more research needs to be done on this subject.

5. Designing New Password Creation Policies

First let us start by defining several terms. An *explicit password creation policy* is any policy that can lay out the exact rules of what constitutes an acceptable password to the user before the user creates their password. Examples of this are minimum length requirements, character type requirements, etc. An *implicit password creation policy* is any policy that has a reject function built into it based on estimated password strength. In addition, this reject function may not be fully apparent to the user until they create their password and submit it to the system. An example of this is a blacklist of "weak" passwords that are not allowed. Finally, an *external password creation policy* is a policy that changes a user's password after it is created in an attempt to add a guaranteed amount of randomness. An example of this would be adding two random digits to the end of a user's password.

Second, it is important to note that the password creation policies discussed in this section are done so in regards to an online password cracking attack. In an offline password cracking attack, it is assumed that an attacker can make hundreds of billions of guesses, if not more. This means an offline password cracking session is a much harder threat to defend against. As an example of that, some of our earlier work cracking the phpbb.com list using two desktop computers can be viewed at [11], where we

achieved over a 97% success rate during the course of a several month cracking session.

As we've discussed in Section 4, while explicit password creation policies can frustrate an attacker and reduce their chances of success, they still leave a system vulnerable to an online password cracking attack due to user behavior. Even more complex rules such as requiring passwords to be 14 or even 21 characters long may be subverted in some cases by users choosing common keyboard combinations, or simply typing the same seven character password in two or three times. Another way of looking at it is that explicit password creation policies do not provide a base level of security by themselves since people tend to follow common patterns.

A very novel and intriguing idea was proposed in [12] to add a limited amount of guaranteed security to explicit password creation policies by randomly selecting a policy when users create their passwords. Rules such as some users would have to include a digit in their password; while other users would be forbidden from using numbers, would be selected with a random chance. Since the attacker would not know which set of rules the user created their password under, they would then have to structure their attack to target multiple creation policies. One potential problem with this, besides user annoyance, is that this approach does not stop a user from selecting a weak password under the random policy. Since the number of policies is finite, an attacker may still be successful by guessing the most common passwords for each policy.

External password creation policies have also been considered by several researchers. The advantage is that they provide a set baseline of guaranteed randomness that is enforced by the system. This guaranteed randomness is imposed by allowing the user to select their base password, and then adding values to it that the user would then have to remember. In a way this can be thought of as a system selected PIN that has to be typed in along with the user's password, though the PIN is incorporated directly into the password. The most extreme example of this is a completely random password assigned to the user. There has been research to make this approach more user friendly, such as assigning the user a random passphrase instead of a random password [13]. Other approaches have attempted to add randomness after the user selects their password by appending or inserting random value to it. The user would then have to remember these system defined mangling rules as well. Perhaps the best study of this tactic has been [14] where researchers attempted to examine user acceptance of such a policy. One interesting point that they found was that as more intrusive creation policies, (where additional random values were inserted into the user's password), were implemented, users started selecting simpler base passwords. It's hard to fault the users for this behavior since there is a limit to what is easily memorable.

The main problem with external password creation policies is that they only function effectively if they are applied to a limited number of logins. This is due to the fact that the system added randomness would change between passwords used at different sites. While this may be seen as a plus, as users would not be able to re-use the exact same password across multiple accounts, the chances of them re-using the same base password remains extremely high. In addition, the main coping strategy for remembering a large number of passwords that were created under external password creation policies would most likely result in a majority of users writing their passwords down. While it has been argued by several security experts that writing down your passwords might not be a negative trend [15], at the very least we lose the flexibility and portability that fully human memorable passwords provide.

This leaves us with implicit password creation policies. Besides being used on their own, implicit password creation policy can also be combined with other password creation policies as well. For example a blacklist may be used along with an explicit creation rule requiring the user to include at least one uppercase letter. Also implicit policies by their very nature may need to be retrained to take into account new user behavior caused by the policy itself. To illustrate this, if the most common passwords are banned, new common passwords may appear. The goal then is to find a reject function that forces users to choose passwords sufficiently different from each other to deter an online attack.

As shown in the experiments in the previous section, a blacklist policy covering at least 50,000 prohibited passwords seems to provide a large degree of security. The biggest issue with a blacklist is that by itself it still may not provide enough security for high value systems. This was demonstrated in Fig 4.3.2 where even though the blacklist provided much more security than any of the other explicit policies, it still had a failure rate of 0.84% when an attacker was allowed 4096 guesses.

One potential solution to this problem is to take the grammar we described in a previous paper [18] and use it to evaluate the probability of user selected passwords. In that paper, we designed a password cracking program that was trained on previously disclosed passwords. Our current version of this cracking program learns information such as the frequency people use certain words, case mangling, basic password structure, the probability of digits and special characters, etc. and uses that information to construct a probabilistic context free grammar that models how people select passwords. Our password cracker then proceeds to make guesses in probability order according to that grammar. In head to head tests with existing password cracking tools, this approach proved to be much more successful at cracking passwords. We originally designed our cracker for law enforcement to help them deal with strong encryption, but we quickly found out that it was also useful for the defender to give an estimate on how strong a password actually was, or at least how different it was from the grammar that the password cracker was trained on. Therefore we can parse new passwords, and assign them a probability according to a previously trained grammar. An example of parsing a password, using a simplified grammar, can be seen in Fig. 5.1.

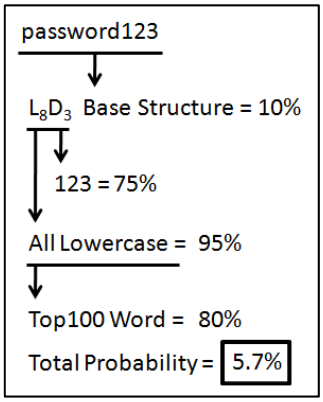

Figure 5.1 Parsing the Password "password123"

Once we have a probability associated with a password we can very easily build a reject function that will disallow any password whose probability falls above a set threshold. What's more, since our underlying system is built on the idea of context free grammars, it becomes simple to select one or more productions and randomly suggest a new replacement that would cause the probability of the password to fall within acceptable limits. If the user does not like any of the suggestions, they are also free to select a new password on their own and resubmit it to the system. An example of this process is shown in Fig. 5.2.

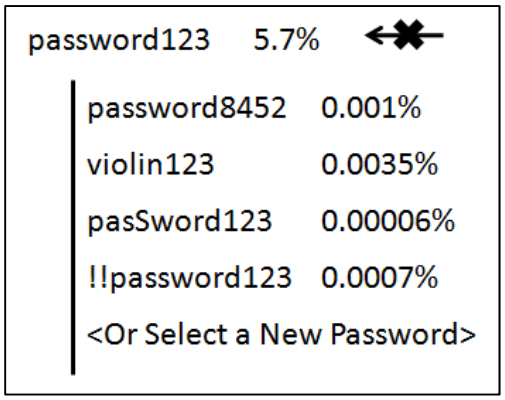

Figure 5.2: Rejecting a Weak Password and Suggesting a Stronger One

Please note: the probability assigned to a password is not the probability of an attacker cracking it, but instead the probability of it being produced by our grammar. The probability threshold would then be set through a series of benchmarks where the remaining passwords productions are unlikely enough that they don't provide an attacker any high probability guesses to make.

There are several advantages to this approach. Almost certainly the most important one is that the probability assigned to the different mangling rules is learned from analyzing real user behavior. This stands in stark contrast to methods like the NIST SP800-63 where they had to decide if uppercasing a letter and adding digits was worth 6 bits vs. 8 bits of entropy. Second, our method assigns a probability score to an individual password vs. an entire password creation policy. This leaves a user free to construct the password however they like as long as their password meets the acceptable probability threshold. For example the user could choose a very long passphrase of all lowercase letters, or have a much shorter password that included digits and capitalization. Furthermore, by displaying the base probabilities of the password, and suggesting new productions, our reject function could help educate users as well about what constitutes a strong password. In addition, by offering new suggestions or giving the user a chance to select an entirely new password, our reject function works very similar to how most websites and online applications recommend new usernames when the selected one has already been taken. This may help drive user acceptance of this implicit password creation policy.

Just like our password cracking program can be retrained to target specific individuals, additional modifications can be made to our grammar to help resist targeted attacks. For example a special dictionary can be designed to match usernames, website names, project names, project acronyms, e-mail addresses, etc and give them a higher probability then they normally would have. This would not stop a user from incorporating them in their password, but it would require them to apply additional non-standard mangling rules for the password to be accepted.

This is merely a suggestion though, and much more research, (such as actual use case studies), needs to be done before such a password creation policy can be considered secure. The main point though, and the reason why we put forward this idea, is to highlight the fact that there are other approaches that can be taken to strengthen the security of user generated passwords besides forcing the users to create longer passwords with more character requirements.

6. Conclusion

Our experiments categorically show that the notion of password entropy, as put forward in the NIST SP800-63 document, does not provide a valid metric for measuring the security provided by password creation policies. This is not to cast dispersions at the rest of the SP800-63 document which is otherwise of the highest quality. Furthermore, we validated the findings in [7], using empirical evidence, that there is no way to convert the notion of Shannon entropy into the guessing entropy of password creation policies.

Moving on from that, we then proceeded to evaluate the security that common password creation policies, such as minimum password length, and character set requirements provide against online attacks. Our findings were that absent an external password creation policy where the system manually adds randomness to a user's password, or an implicit policy where a reject function disallows weak passwords, most common password creation policies remains vulnerable to online attack. This is due to a subset of the users picking easy to guess passwords that still comply with the password creation policy in place, for example "Password!1". Whether such online attacks are feasible or cost effective depends on other factors, such as lockout policy, value of the target, user training etc. By conducting our experiments using real life sets of disclosed passwords though, this provides a much greater insight on how people create passwords and the vulnerability of those passwords to attack.

Finally we put forward several other methods for password creation policies, including our proposed method to evaluate the probability of a human generated password by parsing it with a grammar trained on previously disclosed password lists. This allows us to build a more robust reject function compared to a simple blacklist, while attempting to provide the most user freedom possible, given the security constraints of the system, when selecting their passwords.

More work remains to be done on this topic, as there still are multiple issues that remain unresolved. For example, how do passwords people use on high value targets, such as corporate networks or bank accounts, compare with the passwords collected from various other websites? It is a mixed blessing that the opportunities for this research to be performed in the future are only growing as more sites are compromised and more datasets become public. It is our hope that this paper will expand the discussion on using empirical data collected from non-standard sources to evaluate the security that different policies and technologies provide us.

7. Acknowledgements

This work was supported in part by the U.S. National Institute of Justice under Grant 2006-DN-BX-K007. Also, we would like to acknowledge the

contributions that Breno de Medeiros and Bill Glodek made in the development of the password cracking grammar described in this paper.

8. References

[1] W. Burr, D. Dodson, R. Perlner, W. Polk, S. Gupta, E. Nabbus, “NIST Special Publication 800-63-1 Electronic Authentication Guideline”, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, April, 2006

[2] Office of Management and Budget, “Draft Agency Implementation, Guidance for Homeland Security, Presidential Directive 12”, August 2004.

[3] P. Bowen, A. Johnson, J. Hash, C. Dancy Smith, D. Steinberg, “NIST Special Publication 800-66 An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Security Rule”, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD.

[4] C.E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, July, October, 1948.

[5] C. Herley, “So Long and No Thanks for the Externalities: The Rationl Rejection of Security Advice by Users.” NSPW 09, September 8-11 2009 Oxford, United Kingdom.

[6] A. Vance, “If Your Password is 123456 Just Make it HackMe” New York Times, January 20th, 2010. Page A1.

[7] E. R. Verheul. “Selecting secure passwords”, CT-RSA 2007, Proceedings Volume 4377 of Lecture Notes in Computer Science, pages 49–66. Springer Verlag, Berlin, 2007.

[8] J.L. Massey, “Guessing and Entropy,” Proc. 1994 IEEE International Symposium on Information Theory, 1995, p.329.

[9] The OpenWall Group, [Software] John the Ripper password cracker, [Online Document] [cited 2-19-2010] Available HTTP http://www.openwall.com

[10] A list of popular password cracking wordlists, 2005, [Online Document] [cited 2010 January 14] Available HTTP http://www.outpost9.com/files/WordLists.html

[11] M. Weir and S. Aggarwal. “Cracking 400,000 Passwords or How to Explain to Your Roommate why the Power-Bill is a Little High”, Defcon 17, Las Vegas, NV, August 2009

[12] J. Leversund “The Password Meta Policy” [Online Document] [cited 2010 April 16] Available HTTP http://securitynirvana.blogspot.com/2010/02/password-meta-policy.html

[13] G. Bard, “Spelling-Error Tolerant, Order Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric” Fifth Australasian Symposium on ACSW Frontiers - Volume 68 (Ballarat, Australia, January 30 - February 02, 2007), 117-124.

[14] A. Forget, S. Chiasson, P.C. van Oorschot, R. Biddle, “Improving Text Passwords through Persuasion.” Symposium on Usable Privacy and Security (SOUPS) 2008, July 23–25, 2008, Pittsburgh, PA USA.

[15] B. Schneier, “Write Down Your Password”, June 17, 2005 [Online Document] [cited 2010 April 16] Available HTTP http://www.schneier.com/blog/archives/2005/06/write_down_your.html

[16] Various Authors, “Faithwriters.com hacked message posts” [Online Document] [cited 2010 April 16] Available HTTP http://forums.crosswalk.com/m_4252083/mpage_1/tm.htm

[17] B. Ryan, “The Hacking of the http://db.singles.org” [Online Document] [cited 2010 April 16] Available HTTP http://msmvps.com/blogs/williamryan/archive/2009/02/22/the-hacking-of-http-db-singles-org.aspx

[18] M. Weir, Sudhir Aggarwal, Breno de Medeiros, Bill Glodek, “Password Cracking Using Probabilistic Context Free Grammars,” Proceedings of the 30th IEEE Symposium on Security and Privacy, May 2009.

[19] R. Morris and K. Thompson. “Password security: a case history” Communications. ACM, 22(11):594–597, 1979.

[20] A. Narayanan and V. Shmatikov, Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff, CCS’05, November 7–11, 2005, Alexandria, Virginia

[21] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password Memorability and Security: Empirical Results. IEEE Security and Privacy Magazine, Volume 2, Number 5, pages 25-31, 2004.

[22] T. Wu, "A real-world analysis of kerberos password security," in 1999 Network and Distributed System Security Symposium, February 1999.

[23] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your botnet is my botnet: Analysis of a botnet takeover," Tech. Rep., April 2009.

[24] Sophos, “Security at risk as one third of surfers admit they use the same password for all websites”, [Online Document] [cited 2010 July 14] Available HTTP http://www.sophos.com/pressoffice/news/articles/2009/03/password-security.html

[25] L. Clair, L. Johansen, W. Enck, M. Pirretti, P. Traynor, P. McDaniel and T. Jaeger, “Password Exhaustion: Predicting the End of Password Usefulness” ICISS, volume 4332 of Lecture Notes in Computer Science, pages 37-55, 2006.

[26] J. Bonneau, S. Preibusch, “The Password Thicket: Technical and Market Failures in Human Authentication on the Web”, The Ninth Workshop on the Economics of Information Security, WEIS 2010.

[27] K. Zetter, “Weak Password Brings ‘Happiness’ to Twitter Hacker” [Online Document] [cited 10`0 July 19] Available HTTP http://www.wired.com/threatlevel/2009/01/professed-twitt/

Appendix 1: Information about the Password Lists

The RockYou.com List:

This list was originally obtained by a hacker via a SQL injection against the RockYou.com website [6]. The actual exploit code was first disclosed on the darkc0de.com blackhat message board, where multiple hackers took advantage of it. One of them later publicly posted the list and it is now widely available. Theoretically the RockYou website has a password creation policy requiring passwords to be between 8 and 14 characters long and to NOT include any special characters. This may have been implemented after the attack since the actual list contains many passwords that do not meet those requirements. The RockYou list also includes multiple passwords for various social networking sites such as Facebook, MySpace and Friendster. The list we managed to obtain did not include any usernames or e-mail addresses.

The FaithWriters.com List:

It is unknown how this site was broken into, but most likely it was due to an SQL injection attack [16]. In the list we obtained, both the e-mail address and the password were included. The FaithWriters website was primarily composed of Christian writers. It is suspected that the attacker who hacked the site was associated with the 4chan.org or Ebaumsworld message boards. The faithwriters website had a minimum password length requirement of six characters long, and except for six passwords in the list, all of the passwords complied with it.

The Singles.org List

The singles.org site was broken into via query string injection, (aka all authentication was done via URLs) [17]. The site advertised itself as a dating website for Christian singles. The news of the hack quickly spread to the 4chan.org and ebaumsworld message boards where users there quickly exploited the vulnerability to gain access to all of the passwords on singles.org. In many cases, the malicious attackers then used those passwords to log into other accounts belonging to the users of singles.org. Since many people choose the same password for all of their online accounts, this lead to several serious compromises of Facebook accounts, webmail accounts, Amazon.com accounts, Paypal accounts, etc. The password creation policy of the site required all passwords to be eight or less characters long. This list we obtained contained e-mail addresses and passwords.

The Neopets.com List

It is unknown how this list was originally obtained, but there is a high probability it was done via a phishing attack. Researchers first became aware of this list when it was posted publicly on the pastebin.com website. While it is possible that this list is not associated with the Neopets site, that is unlikely due to the large number of passwords that correspond to common neopets terminology. Neopets.com is website game where users raise electronic pets and battle each other. The game itself is primarily targeted to a younger audience. The password list contained e-mail addresses and passwords.

The PhpBB.com List

The PhpBB list was originally obtained by a hacker who exploited a flaw in a 3rd party plug-in associated with the phpbb bulletin board software [11]. This is ironic since the site itself is the main development website for that bulletin board. The site did not store user passwords in plain text. Instead all of the passwords were hashed using either one round of MD5, or using a salted hash, (consisting of several thousand rounds of MD5). The reason for these two hashing algorithms was that the site had upgraded their forum software, but until a user logged in, they were not converted to the new password hashing scheme. The attacker had attempted to crack a subset of the passwords using an online password cracking program, and managed to crack only 24% of the passwords they targeted. The attacker then proceeded to publish online all of the password hashes, the passwords they managed to crack, along with a write-up of their attack. Since then we have managed to independently crack 97% of all the MD5 hashed passwords from this set using two desktop computers. The PhpBB site did not enforce any password creation policy.

Appendix 2: Statistical Breakdown of the Password Lists

	RockYou.com*	FaithWriters.com	Singles.org	Neopets.com	Phpbb.com**
Number of Passwords	32,603,388 total	6,193	24,870	11,732	259,424
Average Password Length	7.88 characters	7.69 characters	6.62 characters	6.68 characters	7.27 characters
% that Contain Uppercase	5.95	9.43	8.51	2.53	7.21
% that Contain Digits	54.08	43.54	32.88	57.19	45.77
% that Contain Special Chars	3.45	0.14	0.20	1.78	1.33
% that Only Contain Lowercase Letters and Digits	90.76	90.50	91.31	95.61	91.55
% that are 7+ Chars Long, and Contain Uppercase, Lowercase, Digits + Special	0.14	0.03	0	0	0.11

*The RockYou password statistics are taken from the RockYou32 training list which contained 1 million randomly selected passwords
**The Phpbb statistics only include the 97% of passwords we managed to crack