Analyzing Password Strength

Martin M.A. Devillers

Juli 2010

ABSTRACT

Password authentication is still the most used authentication mechanism in today's computer systems. In most systems, the password is set by the user and must adhere to certain password requirements. Additionally, password checkers rank the strength of a password to give the user an indication of how secure their password is. In this paper, we take a look at a large database of user chosen passwords to determine the current state of affairs. In the end, we extract a model from the database and provide our own password checker which ranks passwords in various ways. We ran this checker against our dataset which shows that over 90% of the passwords is highly insecure.

INTRODUCTION

As we perform increasingly important tasks from our living room computer, the topic of computer security also becomes increasingly important. And indeed, many advances have been made in the field of computer security to protect home users from digital crime. For instance, the wireless transmission protocol, Wi-Fi, has had several (much overdue) security overhauls to protect home users from being eavesdropped or worse (1). However, when we look at what we typically use to *authenticate* ourselves with, we are stuck with a system that dates back to the Roman empire (2): passwords.

Although an ancient concept, password authentication is, and most likely will be for a long time, the most used authentication mechanism for computer users.

Password authentication requires no specialized hardware, such as with fingerprint authentication, can be easily implemented by developers and just as easily used by users. In short: It's usable.

But is it safe? The topic of "Security versus Usability" has always been of much debate in the computer security world (3). Users must be protected from harm by security controls, but these controls may not interfere (much) with the tasks the users want to perform. For instance, a firewall that simply blocks all traffic can be considered secure, but heavily impedes the overall usability of the system.

Security experts or system developers are usually the ones who have to make this tradeoff, but with password authentication, this task is essentially passed on to the user (3): One can either choose a short and simple password, which is easy to remember but also easy to crack, or a very long and complicated password, which is hard to remember but also hard to crack.

Unfortunately, most users do not see a tradeoff: They see an obstacle and they will choose the path of least resistance to overcome that obstacle. Thus, short and simple passwords that are easy to remember, but also easy to guess, are used (4).

To stop users from using weak passwords, most systems enforce certain requirements that a password must meet before it gets set. Examples of common requirements are minimal length of the password, the occurrence of uppercase letters, digits and/or symbols in the password and inequality with the user's username or e-mail address (5).

However, holding on to the principle of the path of least resistance, one can expect users to try and 'circumvent' these requirements in a predictable manner (4). For instance, given that an user starts out with an actual word such as 'house' and the requirement that the password must contain at least one digit is given, one might expect the user to simply suffix the word with a single digit. In the findings which we will present, you will see that 15% of the passwords were a word or name suffixed with the number one.

In this paper, we will first discuss the dataset which we used to perform our research. After this, we will show you the results of our preliminary tests on this dataset. In the next chapter, we will go one step further and extract actual patterns from the passwords. After this, we will show you how we used a probabilistic approach to password analysis. In the final chapter, we will present you our password checker, which combines the results of all the previous chapters.

DATASET

On December 4th 2009, a hacker breached a company database of RockYou![1] containing the usernames and *unencrypted* passwords of about 32 million users (3). This database was subsequently published to the internet and is now in wide circulation. Obviously, we don't condone hacking, but the presence of this database gives us an unique opportunity to perform a large scale empirical study on passwords.

The most notable fact about the aforementioned affair would not be that a large database was hacked, but that the passwords inside the database were stored in unencrypted form (so-called plaintext passwords). These days, it is common practice to salt and hash passwords before permanently storing them, which makes it generally hard to study passwords even when granted access to the right databases.

The RockYou! database was acquired in the form of a long text file where each password resides on its own line. The file contained no other information, such as the usernames. Since the source of the data was questionable at best, we ran various tests and filters to ensure the quality of the data. Various noise factors were discovered:

1. Analysis showed that entries longer than 30 characters were more often noise than actual passwords. A lot of entries were pieces of HTML or JavaScript, which might be indicative of an injection attack or some kind of input anomaly. Others contain a single character repeated up to a hundred times. The most likely explanation for this anomaly, are users who want to quickly sign-up and fill in bogus information to get a onetime account. Such entries cannot be considered to be actual passwords, since the user does not have the intention of memorizing or re-using the password. Since these entries are much longer than normal entries, a filter was applied that removes any entry longer than 30 characters.

2. At various points in the database, the character encoding switches. For instance, the first 3 million passwords contain Unicode characters in their HTML encoded form. Hereafter, Unicode characters are stored in their actual form, which might be indicative of the backend switching to UTF-8 encoding support. Similarly, the file itself is part encoded in ANSI and part encoded in UTF-8. Character analysis gave indication when these switches occurred and a program was written to rewrite the database in one uniform encoding scheme (UTF-8).

3. Various (uncommon) passwords occur multiple times in close proximity to each other. This might be indicative of a single user registering multiple accounts. A filter was applied that removes these extra occurrences. Common passwords did not apply to this filter, as they may naturally occur multiple times in close proximity.

PRELIMINARY TESTS

Before we started with the actual analysis, various basic tests were ran to gain some insights into the database. These include a letter frequency analysis, a character type analysis, a length distribution analysis and a common password analysis.

Letter frequency analysis helps us in various ways. Knowing the frequency of each letter gives us the ability to define a more fine-grained metric for measuring password strength. By grading the chance of occur-

[1] RockYou! (originally known as RockMySpace), based in Redwood City, California is a publisher and developer of applications and other social network services. As of December, 2007 it is the most successful widget maker for the Facebook platform in terms of total installations.

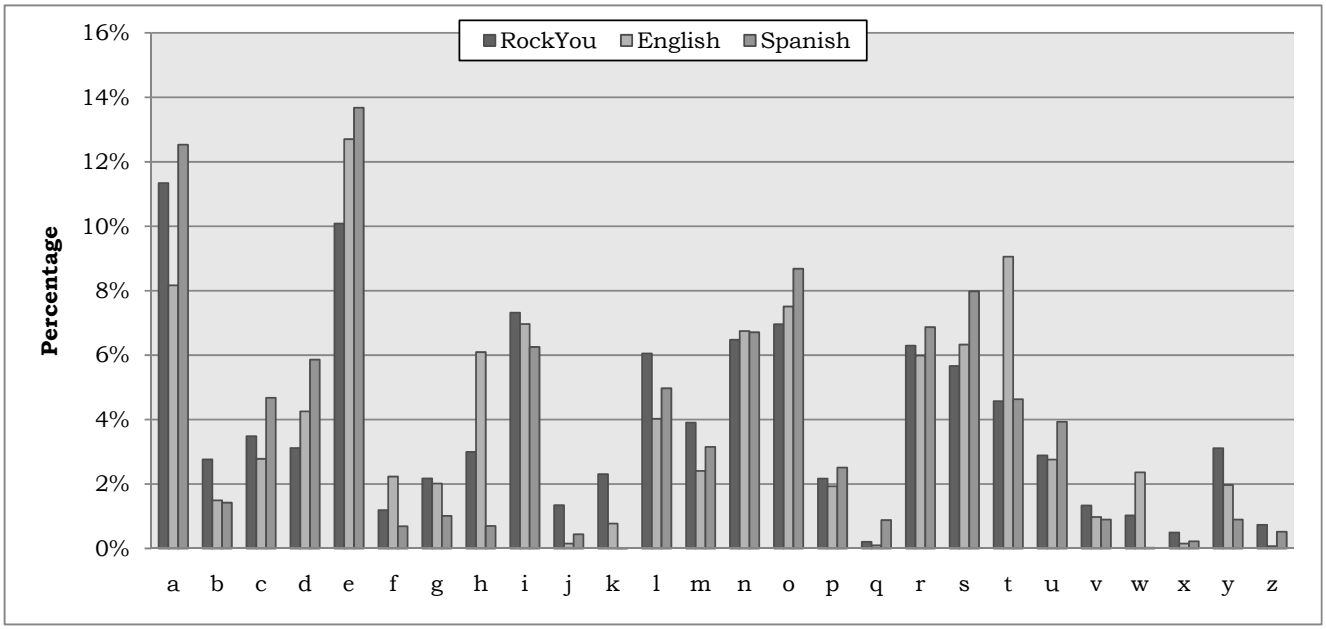

Figure 1 – Letter frequencies of the RockYou! dataset, English and Spanish language

rence of each individual character rather than grading each letter a flat twenty-sixths chance of occurrence.

Furthermore, letter frequency analysis allows us to measure the degree in which the passwords conform to actual words. This helps us to better understand if the dataset follows a language and what language this might be (6).

Figure 1 shows the letter frequencies of the dataset and those of the English and Spanish language. As you can see, the distribution of the RockYou! dataset shows great similarity to those of the English and Spanish language. When we do a quick search for English and Spanish words in the dataset, we find that 10% of the entries match English words and 2.5% match Spanish words.

Character type analysis looks at every password and flags what kind of characters make up the password. We distinguish between the following character types:

- *Lowercase*, which are the standard lowercase letters of the alphabet.
- *Uppercase*, which are the standard uppercase letters of the alphabet.
- *Digits*, which are the digits zero through nine
- *Symbols*, which are any characters found in the non-extended ASCII set that do not belong to the above categories. Most of these can be found on a standard American keyboard.
- *Unicode*, which are any characters that do not belong to the above categories. Examples of these are the euro sign and the Japanese alphabet.

Given these categories, $2^5 = 32$ combinations are possible. However, we expect that only a few will be really prominent.

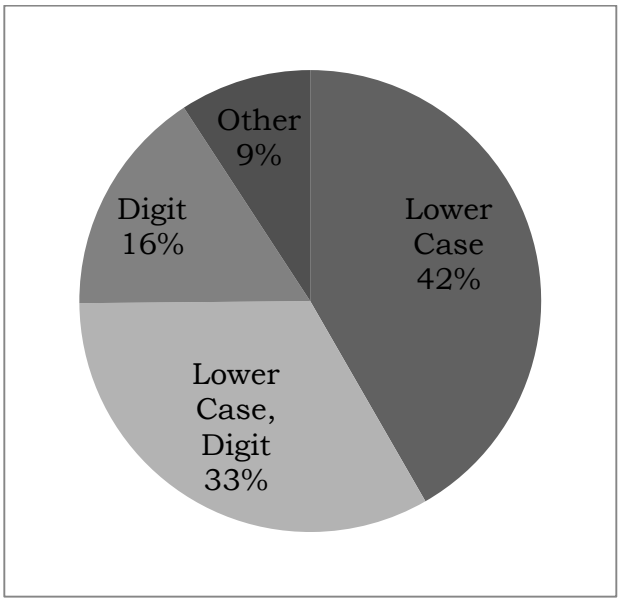

Figure 2 – Character type analysis

Figure 2 shows the results of our character type analysis. The largest category, which nearly makes up for half the dataset, are passwords that consist solely out of lowercase characters. This is troublesome, considering that most passwords are only 6 to 8 characters long and the letters found in the passwords conform to that of a language.

This would suggest that a significant part of our dataset are words or names. Another troublesome result, are the passwords that consist solely out of digits, which represents 16% of the dataset. Numeric passwords have limited complexity.

Length distribution analysis gives us insights in what the common length of user chosen passwords are.

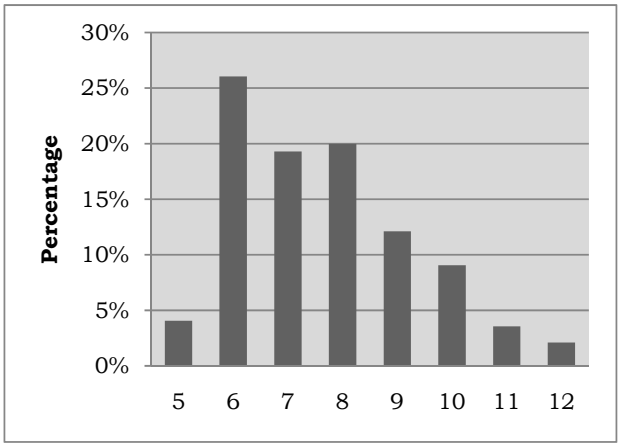

Figure 3 – Password length distribution

The results of the analysis, as shown in Figure 3, do not show a normal form distribution, but rather a truncated form. We argue that this is a result of minimum password length requirements. Strangely, the database contained entries as short as one character. At the moment of writing, the RockYou! website enforces a minimum 8 character password length, but this was most likely less (or non-existent) in the past (10).

In the previous chapter we already noted that we ignored any entry longer than 30 characters. The range of passwords of size 15 through 29 covered about 2 percent of the database. Half the passwords of length 20 and above were e-mail addresses, thus explaining their unusual length.

Finally, we take a look at the most common passwords. We expect these passwords to be really weak, as a password that is widely used is most likely one you can logically guess, find in a common password listing or is context dependent (such as the name of the service)

Password	Count	Percentage
123456	290731	0.8918%
12345	79078	0.2426%
123456789	76790	0.2356%
password	59463	0.1824%
iloveyou	49952	0.1532%
princess	33291	0.1021%
1234567	21727	0.0666%
rockyou	20903	0.0641%
12345678	20553	0.0630%
abc123	16648	0.8918%

Table 1 – Top 10 passwords

Table 1 shows the top 10 passwords with their absolute counts. The use of numerical sequences immediately stands out as 5 of the 10 passwords represents this class.

When we break the numbers down to percentages, two troublesome conclusions can be drawn:

1. Roughly one out of every hundred passwords is the sequence '123456'

2. The top 10, 100, 1,000 and 10,000 passwords cover respectively 2, 5, 11 and 22 percent of all passwords.

In other words, if an attacker would try to gain access to a system by trying the top 10 most common passwords using a listing of known accounts. He could expect to succeed within 25 accounts, costing him only 250 guesses. Given more accounts and just the password '123456', success could be expected with the 50th account, costing only 50 guesses. This attack is feasible without automation and low-profile enough to not raise any alarms.

We compare our results with other top tens of common passwords (4) (5) in Table 2. All highlighted cells represent passwords that are present in our top 10. It should be noted that most other passwords are in our top 100 and all are present in our top 1000.

RockYou!	PCMag	M.Burnet
123456	password	123456
12345	123456	password
123456789	qwerty	12345678
password	abc123	1234
iloveyou	letmein	pussy
princess	monkey	12345
1234567	myspace1	dragon
rockyou	password1	qwerty
12345678	blink182	696969
abc123	*Firstname*	mustang

Table 2 – Top 10 comparison (the last entry of PCMag 'firstname' is a placeholder for an actual firstname)

Although these overlapping lists, strengthens our believe that the RockYou! dataset is representative, it at the same time saddens us to see that so many users still use very predictable passwords.

PATTERN ANALYSIS

The previous chapter showed several ways of defining characteristics of a password. Now, we want to go a step further and extract actual patterns from passwords. This will help us to better understand how passwords are formed and ultimately allow us to construct an improved password checker.

M. Dell'Amico (6) studied a much smaller database of roughly 10,000 entries. Various regular expressions were ran against the database, in an attempt to recognize patterns in passwords. We repeat their experiment on the RockYou! database and present both their and our results below in Table 3

Expression	Example	IIMS	RockYou
[a-z]+	abcdef	51.20%	41.69%
[A-Z]+	ABCDEF	0.29%	1.50%
[A-Za-z]+	AbCdEf	53.74%	44.05%
[0-9]+	123456	9.10%	15.93%
[a-zA-Z0-9]+	A1b2C3	93.43%	96.20%
[a-z]+[0-9]+	abc123	14.51%	27.69%
[a-zA-Z]+[0-9]+	aBc123	16.30%	30.16%
[0-9]+[a-zA-Z]+	123aBc	1.80%	2.75%
[0-9]+[a-z]+	123abc	1.65%	2.53%

Table 3 – Regular expressions

The most notable difference is the use of digits. Most percentages in the IIMS database indicate a reliance on lowercase characters, whilst in the RockYou! database patterns that contain digits are more prominent. The patterns that describe a word that is suffixed by a number are almost double as popular in the RockYou! database. Passwords that are made purely out of digits are also more popular in the RockYou! database.

Based on these results we can argue that the users of RockYou! have been trained to create more secure passwords, most likely by other applications which enforce stricter requirements. One exception would be the passwords that consist purely out of digits. These passwords are used more often in the RockYou! dataset and we consider these kind of passwords to be very insecure, due to their limited complexity.

Besides the aforementioned expressions, we propose our own set of supplemental expressions. We are most interested in passwords that start with letters and end with digits, as they make up for one third of our dataset and regular expressions can help us better understand this set. Therefore, we repeat the regular expression '[a-zA-Z]+[0-9]+' and keep track of all the numbers that are matched. We then look at the most popular numbers and revert these back to regular expressions.

Number	Count	Percentage
1	1476941	16.37%
123	325963	3.61%
2	284354	3.15%
12	213870	2.37%
3	166762	1.85%
13	150069	1.66%
7	147951	1.64%
11	122630	1.36%
5	120376	1.33%
22	107444	1.19%
23	106425	1.18%
01	102756	1.14%
4	101573	1.13%
07	100693	1.12%
21	100370	1.11%
14	95288	1.06%
10	92655	1.03%
06	86495	0.96%
08	86065	0.95%
8	83819	0.93%
15	83708	0.93%
69	81299	0.90%
16	78506	0.87%
6	76798	0.85%
18	71343	0.79%

Table 4 – Top 25 numeric suffixes

Table 4 shows the 25 most used numeric suffixes, which makes up for halve the passwords matched by the regular expression '[a-zA-Z]+[0-9]+'. We expect single digits to be popular suffixes and indeed, the digit '1' covers over a million passwords that use it as a suffix. However, double digit numbers occur twice as often as single digits, which we did not expect. Interestingly, the digit 9 is only present once in the listing (and not even as a single digit number).

Besides the top 25, we present a second list which contains entries longer than 2 characters from the top 100.

Number	Count	Percentage
101	51065	0.57%
1234	49619	0.55%
2007	30731	0.34%
2006	29122	0.32%
666	24317	0.27%
2008	24300	0.27%
12345	20276	0.22%
2005	18694	0.21%
007	18261	0.20%
420	16470	0.18%
123456	15811	0.18%
1994	14288	0.16%
1993	13695	0.15%

Number	Count	Percentage
1992	13609	0.15%
777	13223	0.15%
1995	13149	0.15%
2000	12613	0.14%
111	12426	0.14%
1991	12358	0.14%
1990	11740	0.13%
2004	11466	0.13%
321	11073	0.12%
1989	10940	0.12%
1987	10689	0.12%

Table 5 – Entries from top 100 numeric suffixes with length larger than 2 characters

Table 5 lists these entries. The length of the entries exposes very apparent patterns. All entries of length 4 are years. We argue that the gap between 1994 and 2004 is indicative of users using the current year during sign-up or a birth year. We know that the user base of RockYou! consists largely out of teenagers. Entries larger than 4 represent numeric sequences while entries with length 3 represent repetitive digits or numbers from pop culture such as the number of the beast (666), lucky sevens (777), James Bond's call sign (007) and four-twenty from the cannabis subculture (420).

All two digit numbers are represented within the top 180 entries, which makes any entry in that range have more than 50% chance of falling in this category. We formulate this in a regular expression that matches any two digit number.

Expression	Description
^(1|12|123|1234 |12345|123456)$	Numeric sequence
^[0-9]$	Single digit
^[0-9]{2}$	Double digit
^(19[7-9][0-9] |200[0-9])$	Year between 1970 and 2009

Table 6 – Regular expressions for numeric suffixes

Table 6 shows the patterns described as regular expressions. It should be noted that the order is of importance. For instance, we consider the use of years as a suffix to be more secure than a single digit suffix. Regular expressions and their order will play an important role in our own password checker which we will present later on.

N-GRAM APPROACH

Another approach to analyzing the dataset is by counting the occurrences of small sized character pairs that make up the password. Such an approach delivers a so-called n-gram model, which is one of the approaches used in a paper by M. Dell'Amico (6) to generate passwords. An n-gram model is a type of probabilistic model for predicting the next item in a sequence. n-grams are used in various areas of statistical natural language processing and genetic sequence analysis.

A n-gram model can be useful in two ways:

1. It can be used to measure the likelihood of a password as a product of the relative occurrences of its containing n-grams. This gives us yet another metric to measure passwords.

2. It can be used to generate new passwords, based on the n-gram model. This could form the basis of a new kind of password attack strategy, which is smarter than a brute force approach. (8)

To build a n-gram based model, every possible combination of characters of every password in the dataset must be extracted. This is required to solve the probability term of a n-gram based likelihood calculation, which can be described with the term:

$$Pr(x_i|x_{i-1}, x_{i-2}, \cdots, x_{i-n})$$

To calculate the probabilistic likelihood of a password, the following formula is applied:

$$P_n(\alpha) = \lambda(|\alpha|) \prod_{1 \le i \le |\alpha|} v(\alpha_{i-n+1} \cdots \alpha_i | \alpha_{i-n+1} \cdots \alpha_{i-1})$$

Where n is the size of the n-gram, α is the password, $|\alpha|$ is the length of the password and $\lambda(|\alpha|)$ is the probability of α being that length.

The formula $v(\alpha_{i-n+1} \cdots \alpha_i | \alpha_{i-n+1} \cdots \alpha_{i-1})$ is the conditional probability of the substring $\alpha_{i-n+1} \cdots \alpha_i$ given all possible strings that can be created from $\alpha_{i-n+1} \cdots \alpha_{i-1}$ when appended with a single character. This is an embodiment of the term $Pr(x_i|x_{i-1}, x_{i-2}, \cdots, x_{i-n})$ which is used by the n-gram model to predict x_i based on $x_{i-1}, x_{i-2}, \cdots, x_{i-n}$ and allows us to calculate the probability of a n-gram.

Since we are calculating the product of a series of probabilities, the assumption is made that the collection of occurrences is mutually independent, which means that we assume that the following property of mutual independence holds:

$$Pr\left(\bigcap_{i=1}^{n} x_i\right) = \prod_{i=1}^{n} Pr(x_i)$$

Although we know that this property does *not* actually hold, the assumption that it does is required for us to simplify the problem. Through this independence assumption, our model assumes the Markov property, which enables reasoning and computation with the model that would otherwise be intractable.

The length of the password determines the amount of iterations, which generally means that the longer the password, the more iterations there are and the smaller the overall outcome of the formula will be. This is taken in account, due to the fact that we're explicitly including the probability of the length of the password, as denoted by $\lambda(|\alpha|)$, in our formula.

In the event that $i < n$, which happens in the first $n-1$ iterations of the product, the substring references a non-existing character index of zero or smaller. These references are substituted with the '»' sign to denote the fact that we are dealing with a starting sequence. Thus, we get the convention:

$$\forall i(i \leq 0 | \alpha_i = »)$$

The set of possible characters is determined by the distinct characters which we encounter in the dataset. Thus, we do not include the full Unicode character set by default. Given the previous description of v, the corresponding formula looks like:

$$v(c_1 \cdots c_n | c_1 \cdots c_{n-1}) = \frac{\sigma(c_1 \cdots c_n)}{\sum_{\bar{c} \in C} \sigma(c_1 \cdots c_{n-1}\bar{c})}$$

The formula σ denotes the number of occurrences of the input string inside the model. The numerator part of the fraction represents the number of occurrences of the string that we are interested in (the n-gram), whilst the denominator represents the number of occurrences of all possible permutations of that string. In the denominator, $\bar{c} \in C$ indicates a character from the set of all possible characters as encountered in our dataset.

The actual value of n has a huge impact on the behavior of the model. In the next few paragraphs, we will discuss various values of n and their implications.

Using a value of one for n (unigrams), will make the algorithm contextless, as the algorithm will only look at the frequency of single letters. This is synonymous to using the character frequency analysis from the previous chapter to grade or predict passwords.

If one were to choose a very large n, then the algorithm would lose its ability to dissect passwords and simply grade a password by its entire string of characters. This is synonymous to calculating the frequency of a given password within the original dataset. A password generator with such a model would (almost) never generate any passwords outside the dataset.

On another note, a large n would also impose problems of the computational kind. As the value of n increases, the amount of possible n-grams increases exponentially. Given a dataset of our size and a n of 5 our algorithm used up nearly a gigabyte of system memory. For a larger n we had to resort to using a smaller subset of our dataset.

Finally, a large n also imposes another problem with the effectiveness of our algorithm. As we said earlier, one of the powerful features of using n-grams is that they can be used to analyze or produce new (unseen) data based on known data.

However, this imposes *some* requirements on the known data. Say we want to calculate the strength of the phrase "password" with $n = 4$, thus giving the formula P_4("password"). This would mean calculating the product of the probabilistic frequencies of all 4-grams contained within the phrase "password". Consider the 4-gram "word", which requires the calculation as shown at the top of this page.

Now, consider a model which has never seen the 4-gram "word". This would result in o("word") $= 0$ and thus v("word"|"wor") $= 0$. As we're taking the product of v in our algorithm this would ultimately lead to P_4("password") $= 0$.

One might argue that this is favorable, since we cannot grade something when we have no background data. However, given the previous example, we *do* have actual values for v for all the 4-grams preceding "word". Thus there is certainly some meas-

ure of occurrence present in the calculation, which is lost when we multiply this by zero. We overcame this limitation by altering the algorithm to raise all occurrences by one and counting any item that never occurs as one instead of zero. This has little impact on n-grams that occur many times and makes our algorithm more robust when dealing with unseen data. The downside being that n-grams that occur very few times will have a less accurate representation.

We used two third of our dataset as a training set for our algorithm and the remaining one third as our test set.

N=1	N=2	N=3	N=4
aaaaa	123456	123456	123456
eeeee	123123	1234567	1234567
ale	12345	12345	12345678
aanaa	121234	12345678	123456789
aaaan	112345	234567	passwo
aaaaaa	123234	milove	12345
aaeaaa	123412	112345	passwor
aeaeae	123450	lover	ilovey
ann	123452	012345	password

Table 7 – Most frequent passwords

Table 7 shows the top 10 most likely passwords as graded by our algorithm for various sizes of n. As we noted earlier, an analysis based on unigrams (1-gram), is basically a character frequency analysis. As vowels are the most frequently occurring characters in the natural language, passwords that contain vowels are graded higher, than those that do not. When we look further down the list we see mostly short pronounceable passwords.

When using bigrams (2-grams), numbers take the upper hand as they form the most frequent two character pair in the dataset. As we've seen from the previous chapter, there are simply a lot of passwords that consist out of or contain numerical sequences. Moreover, the chance that some digit follows another is larger than the chance that some letter follows the other, simply because there are only 10 digits as opposed to 26 letters.

When using trigrams (3-grams), text based passwords start to show up, but numerical sequences still prevail. This top 10 however shows similarities to the actual top 10 of most occurring passwords in the RockYou! Dataset.

Finally, when using quadrigrams (4-grams), numerical sequences are overly present, but the larger list of results show a great deal of overlap with our own top 1000.

From our findings we concluded that using trigrams was a good trade-off between accuracy and usability (in terms of performance).

PASSWORD CHECKER

The previous chapters illustrated various ways in which we can grade password strength. In this chapter, we combine these metrics into a program that can help users to assess their passwords safety. Each step employs one or more metrics, which leads to one or more results. Each result has a name, a short description in natural language what the result represents and a severity level to indicate the impact of the result. Some results are pure informative and therefore have no severity impact.

At the core, our checker does various things.

1. The length of the password is analyzed, as this is a simple metric that can rule a password out as weak without even looking at the actual content of the password.

2. The password is ran across a common list of passwords. These are lists acquired from the internet or calculated by ourselves from our own dataset.

Pattern analysis is employed to determine if the password fits a common pattern. One of the most important parts of this step is to actually break up the password in letter(s), digit(s) and symbol(s). If pattern analysis fails to find a suitable pattern, an unmangling strategy is applied. Some users try to add complexity to passwords by replacing certain characters with digits or symbols (i.e. password becomes p4$$w0rd) (14). This inhibits regular expression based pattern matching from finding actual patterns. The unmangling strategy attempts to revert said mangled passwords to their letter based form. Once a matching pattern has been found, the algorithm looks at what groups the pattern captured.

a. If any words were found during the pattern analysis step, they will be held against common dictionaries of passwords, words and names. We prefer to use small dictionaries of a specific type, since this gives us more power and certainty in describing what the word represents.

b. If any numbers were found during the pattern analysis step, they are inspected to see if they fit a common form (such as a year).

c. If any symbols were found during the pattern analysis step, they are inspected for common occurrence.

3. Finally, we use a 3-gram model based on our dataset to grade the password. To make our 3-gram based algorithm suitable for password grading, we need to map the range of possible values from our algorithm to a set of gradations. First off, the actual range of $P_3(\alpha)$ lies between 1 and 10^{-28}, where the lower limit is a constraint set by our computer environment. We determined the boundary values for the gradations by manually checking the results of our algorithm on several sets of passwords. For instance, running the algorithm on the top 1000 we computed earlier, gave us a good boundary value for passwords that are 'Critical'.

When all results are calculated, they are presented to the user in order of severity. Each result is accompanied with a short message describing the 'problem' that was found.

We used two third of the RockYou! dataset as the training set and the remainder one third as the test set. From the training set, we constructed the various metrics required by our password checker such as the n-gram model and the length distribution. We then ran the checker on our test set. 50% percent of the passwords were reported as being 'Critical' which means such a password should never be used. 45% percent was reported as being 'High', which indicates that the password should be changed or only used for none-critical purposes. Only 5% percent had a severity of 'Medium' or lower.

CONCLUSION

Summarizing, when we look at the RockYou! Dataset, the preliminary tests indicate that passwords are generally short, conform to existing language patterns and show a great deal of overlap. In other words, a significant part of the user base has an insecure password.

Moving onto pattern analysis, we have shown that users that do try to enhance their password, do so in a very predictable manner: Over a million users chose a password consisting out of a word suffixed by the digit one. We extrapolated common patterns from the dataset and presented their occurrences.

The results from our n-gram approach showed how probability calculation can help us assess password strength. We looked at various values of n for determining the optimal size of the model and determined that trigrams (3-grams) were best suited for grading passwords.

The above findings were incorporated in our own password checker, which we then ran against our dataset. The results were, again, troublesome, as 95% of the passwords were marked as being highly unsafe. One mitigating argument is that RockYou! provides a very simple non-critical service, thus users may not feel inclined to provide a strong password. Yet, with halve a million users choosing a short numerical sequence as a password, one may wonder if security in terms of password authentication is at all present.

REFERENCES

1. **Alliance, Wi-Fi.** Wi-Fi Protected Access: Strong, standards-based, interoperable security for today's Wi-Fi networks. *Wi-Fi Alliance.* [Online] April 29, 2003. http://www.wi-fi.org/files/wp_8_WPA%20Security_4-29-03.pdf.

2. **Paton, W.R.** *The histories of Polybius / with an English translation by W.R. Paton.* s.l. : Heinemann ; Harvard University Press, 1922.

3. *Security and usability: the case of the user authentication methods.* **Braz, Christina and Robert, Jean-Marc.** s.l. : ACM, 2006. pp. 199--203.

4. *Users are not the enemy.* **Adams, Anne and Sasse, Martina A.** s.l. : ACM, 1999, Commun. ACM, Vol. 42, pp. 40--46. 0001-0782.

5. *A large-scale study of web password habits.* **Florencio, Dinei and Herley, Cormac.** s.l. : ACM, 2007. pp. 657--666.

6. **Microsoft.** Create strong passwords. *Microsoft Online Safety.* [Online] October 4, 2009. [Cited: November 19, 2009.] http://www.microsoft.com/protect/fraud/passwords/create.aspx.

7. *Human selection of mnemonic phrase-based passwords.* **Kuo, Cynthia,**

Romanosky, Sasha and Cranor, Lorrie F. s.l. : ACM, 2006. pp. 67--78.
8. **Siegler, M.G.** One Of The 32 Million With A RockYou Account? You May Want To Change All Your Passwords. Like Now. *TechCrunch.* [Online] TechCrunch, December 14, 2009. [Cited: Februari 16, 2010.] http://techcrunch.com/2009/12/14/rockyou-hacked/.
9. *Prediction and Entropy of Printed English.* **Shannon, C. E.** 30, 1951, Bell System Technical Journal, pp. 50-64.
10. RockYou Hack: From Bad To Worse. *TechCrunch.* [Online] December 14, 2009. http://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/.
11. 10 Most Common Passwords. *PCMag.* [Online] May 2007. http://www.pcmag.com/article2/0,2817,2113976,00.asp.
12. **Burnett, Mark and Kleiman, Dave.** *Perfect Passwords: Selection, Protection, Authentication.* Rockland, MA : Syngress, 2005.
13. *Measuring Password Strength: An Empirical Analysis.* **Dell'amico, Matteo, Michiardi, Pietro and Roudier, Yves.** Jul 20, 2009.
14. *Fast dictionary attacks on passwords using time-space tradeoff.* **Narayanan, Arvind and Shmatikov, Vitaly.** s.l. : ACM, 2005. pp. 364--372.
15. *Improving text passwords through persuasion.* **Forget, Alain, et al.** s.l. : ACM, 2008. pp. 1--12.