A state-of-the-art password strength analysis demonstrator

by

Nico van Heijningen (0821976)

CMI-Program *Technical Informatics* – Rotterdam University

June 26, 2013

First supervisor	*Dhr. L. van de Zeeuw*
Second supervisor	*Dhr. M.S. Bargh*

Abstract

Due to recent developments: leaks of large lists of user passwords (e.g. LinkedIn), new probabilistic password cracking techniques and the introduction of password cracking using GPUs. Passwords can now be cracked faster than ever before. The leaked password lists have been analyzed by hackers and common patterns found inside the passwords are being exploited to crack others. We have analyzed a collection of these leaked password lists and generated a list of the most common patterns in a probabilistic order furthermore, we compared the distribution of characters inside passwords to that of English text. Next we have built a state-of-the-art password strength analysis demonstrator that is able to show which of these common patterns are contained inside a password and why it could be considered a 'weak' password. The demonstrator is modeled after the realistic scenario of an automated password cracking attack and passwords that assessed 'strong' should therefore 'survive' such an attack. We are convinced our demonstrator is an improvement over the current password strength measurements because it results in a lesser 'false sense of security' amongst its users and helps them make their passwords more resistant against such attacks.

Contents

Abstract **ii**

Introduction **1**

Background **2**

Related Work **5**

Research – Password Patterns **7**

Methodology 7

Results 8

Engineering - Password Cracking Setup **15**

Design - Password strength analysis demonstrator **19**

Conclusions **21**

Recommendations **22**

References **26**

Evaluation **27**

A dictclean.php **28**

B Dataset remarks and numerals **31**

C Pack - 62000 **34**

D Pack - Battlefield Heroes Beta **36**

E Pack - Bitcoin **38**

F Pack - Gamigo **40**

G Pack - Linkedin **42**

H Pack - phpBB **44**

I Pack - RockYou **46**

J Pack - Rootkit.com **48**

K Pack - Sony **50**

L Pack - Yahoo **52**

M Cracking System Benchmarks **54**

N Masks only occurring in one dataset **55**

O Masks occurring in more than one dataset **56**

P Masks occurring in more than one dataset - without three largest masks **59**

Q Comparison of systems **62**

R MD5 pseudocode **64**

S Hashcat’s default bruteforce mask increment mode **65**

T Per position letter frequency heatmaps **67**

Introduction

Text-based passwords play an integral part in the day to day protection of our online identity, a strong password therefore plays an essential role in preventing online identity theft and alike. When registering a new account, password strength meters provide an indication of the strength of the chosen password. The meter supplies a user with feedback whether his or her password is considered 'weak' or 'strong'. The impact of password strength meters on user password selection behavior has been the subject of several studies [1, 2]. These studies suggest that such meters can result in stronger user passwords. Yet a large portion of password meters define password strength as a product of the length of the password and the size of the character set used. They therefore assign to each possible password the same probability, an equation which only holds for completely random passwords, which is definitely not the case for user-created passwords. This suggests that the current implementation of password meters results in a false sense of security.

The strength of a password depends on the time it would take an attacker to correctly guess the password. This, in turn, is dependent on two more abstract defined factors: the number of guesses it takes to find the correct password and the speed of checking the validity of each guessed password. Both factors are influenced by a myriad of psychological and technical factors. Recent developments have had a large influence on these figures: the introduction of probabilistic guessing techniques based on large lists of leaked passwords [6] and the ability to exploit the parallelization of graphical processing units (GPUs) [7, 8].

In this work, we discuss different probabilistic techniques used in password cracking tools, we define the patterns present in large collections of leaked user passwords and we discuss different GPU-based password cracking systems including our implementation of a password cracking setup. Together this results in an improved password strength analysis demonstrator based on state-of-the-art password cracking technologies, tools and methods. Our hope is that this encourages users to choose stronger passwords, leads to more security awareness regarding passwords and leads to a better resistance of users' passwords against automated password cracking attacks.

Background

In this section we will globally sketch the history of the constant battle between password security and password cracking. This section is intended to help the reader grasp how password authentication has become what it is today. Furthermore it introduces several techniques and methods used in the rest of this thesis.

Since Roman times passwords have been used for the purpose of authentication. For a person can be assumed to be who he claims to be when the secret (i.e. password) shared between the authenticator and authenticatee is correct. The ease and low cost of implementation have contributed to password authentication being the most widespread and dominant authentication method for computer systems.

One of the first implementations was used to differentiate between users of time-sharing computer systems. It consisted of a password file containing the actual or *plain text passwords* of all the users. This quickly led to undesirable disclosures of users' passwords: the file could be either *accidentally exposed or purposely stolen* with more illicit intentions. A person or attacker that had obtained such a file could directly login to all the user accounts, which is off course highly undesirable. That is why a technique was devised to –instead of saving the actual password– save a 'fingerprint' or hash of the password [9]. Several of such one-way cryptographic hash functions have been devised to quickly generate a fixed-size bit string when inputted with arbitrary data. These algorithms are designed to be irreversible, it should not be possible to calulate the original data from the hash. Furthermore each returned hash should be unique for the inputted data, any change in input data should result in a change of the hash value. Finally no two pieces of input data should result in the same hash. A good cryptographic hash function meets these requirements yet several older algorithms exist that are prone to attacks [10, 11]. By saving the *hashed passwords* instead of the actual plain text passwords the attacker is not able to log into any of the user accounts directly.

Yet what the attacker can do is simply check if any of the passwords he inputs into the hashing function results in a hash identical to one of the hashes in the password file. If so, the inputted password is identical to the password of the user and the attacker will be able to login to that account. For an attacker to correctly guess a user's password he might have to try a lot of different passwords; in terms of crypto analysis this is called a *key search*. But *human habits* make such a key search a great deal easier. When given free choice most users will choose very short and simple passwords. For example, a word from the English dictionary could be such a simple password. This is why attackers have automated such attacks to try common dictionaries or wordlists called a *dictionary attack*.

To make it harder for an attacker to correctly guess a password, the passwords had to be made less predictable. Instead of using simple words users were encouraged to use gibberish of combinations of words and numbers. Yet because of the increase in computing power the time it took for each hash to be calculated decreased significantly and attackers automated their attacks to just try every possible password. This method of *brute forcing* is therefore called an *exhaustive key search* and is sure to reveal all of the passwords but becomes infeasible at bigger password lengths. Next people were required to satisfy a minimum password length, this is the idea behind a *password policy* to make passwords less predictable and the number of passwords to be guessed incredibly large. The number of possible passwords to guess is called the *key space* and it increases significantly with every character and kind of character that is added. For example, instead of letting a user choose a password of six lowercase letters, making a user choose a nine character password consisting of lowercase letters, uppercase letters and digits will increase the possible key space by a factor of forty million. Because of this increase, the size of the dictionaries and time to calculate all of the possible hashes became too large to be feasible.

Therefore a time-memory tradeoff was devised, by distributedly creating a pre-computed table of hashed passwords. All the attacker would have to do is look up a hash and find the associated password. The so-called *rainbow tables* [12, 13] do not contain all possible hashes rather only the starting and ending position of chains of hashes. By chaining the hashes together the space needed for saving all of the possible hashes is significantly reduced yet it is still possible to calculate all of the intermediate hashes at the cost of some computing time. To counter these rainbow tables a large random value or *salt* was concatenated with each password, increasing the key space with many more orders of magnitudes, making rainbow tables infeasibly large to calculate. Adding a salt to a password does not increase the task of calculating an individual password yet it does not only prevent rainbow table attacks but furthermore it makes each and every password hash unique. Without a salt two identical passwords would result in an equal hash, giving an attacker some knowledge about password reuse. Also an attacker could not check one hash on a whole password file anymore. The calculation of the hash needs to be done for every hash individually increasing the time to crack a password.

Thanks to *Moore's law*, attackers have always been in the most comfortable position. They would only have to wait for hardware to get faster and consequently for their password cracking algorithms to get faster. A recent development is the introduction of video cards or GPUs into the password cracking game. Due to computer games becoming increasingly popular and consumers demanding ever-greater graphics in those games, there have been rapid developments in the hardware making those good looking games possible. This hardware has become accessible to much more general-users thanks to a steady decline in prices. And due to the nature of graphic calculations, the GPU excels in doing a lot of similar small tasks in parallel. As non-game software developers tried to use this existing parallelism to their advantage, GPU-vendors have made their hardware more easily accessible for those developers by creating interfaces to access their hardware. This technique has been labeled 'General-Purpose computing on Graphics Processing Units' (GPGPU). It is now being used to exploit the embarrassing parallelism of standard cryptographic hash functions.

This has been counteracted by the introduction of new hashing algorithms like bcrypt [16] and later scrypt [17], which have been especially designed for the purpose of password hashing. These hashing algorithms have a variable *work factor* to adjust the effort required to calculate the hash. It can be adjusted as time progresses, keeping the time and resources needed to calculate a single hash constant. This brings us to the present day, now attackers are focusing much more on 'smarter algorithms' which once again exploit people's password selection habits. Instead of trying to brute force the whole key space they try to make educated guesses about people's passwords.

Examples of tools that incorporate such known patterns are Hashcat [14] and John the Ripper [15], they have incorporated several techniques to make their password guesses mimic the user habits of password creation. The two techniques we will be focusing on are *masks* and *markov*. The first tries to mimic the common patterns that are present inside passwords (e.g. passwords tend to start with capital letters). The second is a result of the distribution of characters within passwords. Just like English text some characters are used more often than others and some characters are followed more often by specific characters than others (e.g. a ’q’ is rarely used and when used it is nearly always followed by a ’u’).

There are several other –sometimes simpler but still– very effective attacks available. Examples could be the use of ’smart’ dictionary or wordlist attacks: the generation of password guesses by permutating dictionary words (e.g. ’password’, ’p@ssword’, ’Password’, ’passw0rd’), combining dictionary words (e.g. the passwords ’dog’ and ’cat’ becomes ’dogcat’ and ’catdog’). These attacks use user-defined rules to know what part of the password to permutate. A example would be the permutation of l33t-speak, often people are adviced to substitute the letter ’e’ with the number ’3’ and the letter ’o’ with the number ’0’ etcetera. This pattern can be exploited by such attacks called *rule-based attacks*. We will use these attacks in our implementation yet won’t discuss them nor explain them as these attacks themselves are again based on the exploitation of patterns inside passwords.

Related Work

In this section we outline the current state of implementation of password meters. We discuss research performed by others related to ours, we discuss in which ways we build upon other studies and the ways in which we differ from those studies.

To give a better insight into the current state of implementation of password meters we have dissected the password meters from Microsoft [3] and Intel [4] in order to discuss how these password meters define password strength. Both password meters are implemented to run client-side as a Javascript script, therefore no information regarding the password or password strength is sent to either Microsoft or Intel. We will now highlight the most important parts of both algorithms followed by a rewritten mathematical abstraction of the algorithm.

The password meter from Microsoft defines password strength as:

```
var bits = Math.log(charset) * (passWord.length / Math.log(2));
```

$$|\log_2 (\frac{1}{charset^{length}})| = password\,strength$$

The password meter from Intel defines password strength as:

```
var entropy = Math.pow(lowercase_bits ,lowercase_chars) * Math.pow(
    uppercase_bits ,uppercase_chars) * Math.pow(digit_bits ,digit_chars) * Math.
    pow(special_bits ,special_chars);
var entropy = entropy / 2;
var _std_comp_power = 2 * Math.pow(2, 33);
var hours = entropy / _std_comp_power;
```

$$\left(\frac{\left(\frac{charset^{length}}{2}\right)}{computer\,speed}\right) = time\,to\,crack\,the\,password$$

Note both the source code examples use the definition of entropy an bits, these definitions are not equivalent to the information entropy as defined by Claude Shannon. There are some similarities yet these are beyond the scope of this work.

The most important thing to note from the mathematical abstractions is the use of $charset^{length}$ to calculate the key space. Both algorithms assign to all possible passwords in the key space the same probability; they do not distinct between the password 'password' and the password 'fsejslkq', which might be used much less.[1]. Both password meters come with a disclaimer similar to "*Note: This does not guarantee the security of the password. This is for your personal reference only.*". Such a disclaimer is a requirement as the security of a password depends on many more factors than its strength only. Yet the meter results in a 'false sense of security' as the definition of the password strength by both meters is not up to date with the latest password cracking techniques the definition by the password meters is more like a shot in the dark than an educated guess.

This problem was the motivation for Weir et al. [5, 6] to develop an algorithm which generates probabilistically optimized password guesses. It abstracts a password into its character sets and associates each step with a probability derived from a training set. Such training sets can be any set of words, from an English dictionary to previously cracked passwords. This results in a probability ranked guessing of passwords, instead of sequentially guessing a six character lowercase password from 'aaaaaa' to 'zzzzzz'. The password 'monkey' will have a higher probability than the password 'aaaaaa' and therefore be tried earlier in the cracking process. A more advanced pattern could be for example, three lowercase letters followed by a digit and finally a number (e.g. 'dog1@'). Here the different characters sets will be abstracted as such

$$dog1@ \rightarrow L_3D_1S_1$$

This pattern ($L_3D_1S_1$) learned from the training set can then be used to generate new password guesses

$$S \rightarrow L_3D_1S_1 \rightarrow L_34S_1 \rightarrow L_34! \rightarrow cat4!$$

The masks used by the Hashcat tool have a large resemblance to this abstracting technique and this is why we chose to use them. Gage, et al. [18] have conducted a user study quantifying different password strength estimates under different password-composition-policies. They have implemented a distributed technique for determining the number of guesses needed to find a specific password, implementing several probabalistic password guessing algorithms including the one defined by Weir et al. [6]. They used a training set of 12 thousand purposely collected passwords which were created under different password policies, and showed that a blacklist using state-of-the-art password guessing techniques is very effective in letting people choose stronger passwords than a simple dictionary blacklist as, for example, implemented by Twitter. Yet their research has not resulted in any practical implementation as our work is intended to do.

In his thesis Chrysanthou Yiannis [19] developed a fast, dynamic and self-adjusting attack on leaked password lists. The attack iterates through different types of password attacks and uses the output of the previous attack as the input for the next. By doing this an automated targeted attack is achieved without the need of any human interaction. This resulted in the cracking of 76% of the passwords of the publicly leaked phpBB list [24]. We consider these password 'weak' as they can be cracked quite simply. Therefore the remaining 24% of not-cracked hashes can be considered 'strong' passwords. We aim to make our password strength analysis algorithm only approve such 'strong' passwords. This is why we have modeled our algorithm to simulate a similar automated password cracking attack.

[1]The Intel password meter does check whether the password is part of a list of 10,000 most commonly used passwords, if so the password is assigned a negligible strength.

Research – Password Patterns

In this section we will first explain how we have gathered and organized the data used in our research. Second we disscuss the experiments we have run against this data followed with their results.

During our literature study we found several statements suggesting common patterns present inside passwords. For example, *"A vast majority of people still follow common patterns, from capitalizing the first letter of their password to putting numbers at the end."* [6]. These statements are based on personal experiences of password crackers, yet we were not able to find any specific figures to back these statements up. This is why we decided to study such common patterns ourselves and define a list of patterns ordered by their occurrence.

Methodology

Several large sets of plaintext and hashed passwords can be found online. Most passwords are leaked by hackers that have broken into a system and have disclosed the user database. For example, the hashes of six million LinkedIn passwords were uploaded to a forum dedicated to password cracking [25]. For each leaked list it is not only the contents that differ, but most lists are organized in different ways and might be saved in different file formats. Furthermore the list could contain additional data not useful for our research purposes. This data possibly consist of: commentary by the hackers, email addresses, usernames, identification numbers or anything else saved in the disclosed database. Therefore we need to parse every single list differently, as we are only interested in the plaintext passwords. The global process of parsing a dataset is as follows:

1. Remove everything but the plaintext passwords
2. Make sure the encoding of the passwords is correct
3. Remove all empty lines
4. Make sure the file uses only Unix line endings (linefeed only, instead of carriage return and linefeed)
5. Create a sorted version

Because of the sheer number of passwords to be processed and parsed we have automated this process quite a bit. We made thankful use of the standard Unix core utilities, like cat, cut, grep, wc, sort and dos2unix to normalize the original leaked password lists. A script was used to check if each password was valid UTF-8 encoded [42].

Yet the process of parsing was still somewhat cumbersome because of each list's properties and background. This is why we would like to thank Matt Weir for providing us with a part of his cracked password list which he used in his research. The table in appendix B [43] includes remarks regarding each dataset's background and properties. Furthermore the percentage of usable passwords per dataset is included. It is good to note that four out of the ten datasets were leaked as plaintext passwords instead of hashed passwords [2].

Consequently all the passwords that were leaked plaintext have been used [3]. The six remaining datasets consisted of hash values only. Therefore the hashes needed to be cracked first, which introduced a bias as the easy passwords will be cracked first and the stronger passwords might not be cracked. This may result in an analysis lacking the stronger passwords of those lists, that is why the table in [43] contains the percentage of the list used in our research. Additionally the table contains the percentage of unique passwords inside the list. We chose to perform our analysis on the datasets containing duplicate passwords as we could not differentiate between duplicate accounts or passwords used by multiple users. This too leads to a possible bias as a lot of duplicate accounts would increase the probability of a pattern.

The majority of the passwords we use in our analysis have already been cracked by other password crackers. The 'Battlefield Heroes Beta' list was the only list which was not, we cracked this list using 'stupid' brute force methods over a weekend. We were able to crack over fifty percent of the list before ending the cracking session. We use the list to test the more advanced password cracking methods further on in this thesis.

Furthermore appendices C through L show automatically generated reports regarding the dataset's: line count, mask, character set, and advanced mask statistics. These were generated using the *dicstat.py* script from PACK or Password Analysis and Cracking Kit developed by Peter Kacherginsky [26].

After parsing the cracked and plaintext datasets, we wanted to be able to quickly and easily query the different datasets. This is why we imported every single password from all the datasets into a MySQL database. The first time round we were unaware of the fact that MySQL uses a case insensitive collation by default so the whole process had to be redone, this time using a case sensitive collation [27]. This resulted in a database of more than 45 million passwords and around 3.6 GB in size.

Results

By dissecting Weir's afore mentioned statement we defined three possible common password patterns: the first character of a password is more likely to be a capital, the last character of a password is more likely to be a number and the combination of the former two patterns. Due to time constrains we have analyzed the password patterns by devising more specific experiments.

To generate an answer for these experiments we compute an individual answer for every dataset and take the mean as a representative number for the collection of datasets. We are aware of the possible bias that this method introduces as every dataset is weighed the same. But due to the different properties of each dataset it would be impractical to assign a weight to each one separately. We consider the number of datasets and therefore the sample size large enough to subvert the lack of individual weights. Still, this is important to keep in mind, as the results of the

[2]denoted in [43] as having no hashing function

[3]Incorrectly encoded passwords have not been used that is why some of the plaintext leaks are not used as a whole.

experiments will only be valid for this collection as a whole and not for any individual dataset. To help interpret the data more validly we have calculated the standard deviation of the mean for every answer and denoted this by σ.

Experiment 1 *Is the first character of a password more often capitalized than the second character? (e.g. "Password" vs. "pAssword")*

Yes, of the 10.79% of all passwords containing a capital letter 78.78% has a capital letter at the first position and 27.30% has a capital letter at the second position. The difference in occurrence between the first and second position is 51.48 percentage points (σ=12.49). Figure 1 shows the results distinct by dataset including the average results between the datasets.

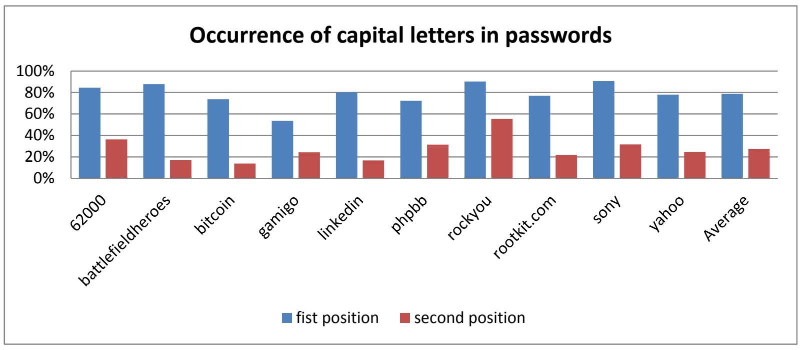


Figure 1: Experiment 1, capital letters

Experiment 2 *Is the last character of a password more often a number than the penultimate character? (e.g. "password1" vs. "passwor1d")*

Yes, of the 62.14% of all passwords containing a number 76.65% has a number at the last position and 67.25% has a number at the penultimate position. The average difference in occurrence of numbers between the last and second to last position is 9.40 percentage points (σ=6.22). Figure 2 shows the results distinct by dataset including the average results between the datasets. It is interesting to note that the 'Gamino' dataset is the only dataset this pattern does not apply to, we were not able to find an obvious explanation for this yet in appendix B [43] some more strange properties of the 'Gamigo' dataset are noted.

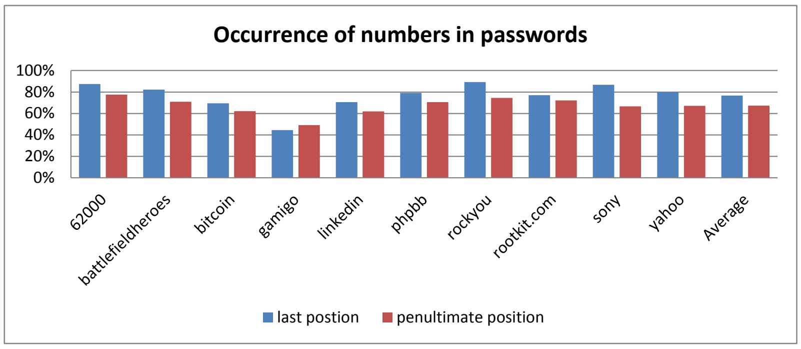


Figure 2: Experiment 2, numbers

Now we combine the first two experiments, creating a more advanced password pattern. Instead of a password just starting with a capital letter or ending with a number, we want to know the difference between the pattern of a password starting with a capital letter and ending with a number and the pattern of a password starting with a capital letter and ending with a symbol.

Experiment 3 *Do passwords tend to start more often with a capital letter and end with a number as opposed to starting with a capital letter and ending with a symbol? (e.g. "Password1" vs. "Password&")*

Yes, of the 8.23% of all passwords containing a capital letter and a number 57.53% start with a capital letter and end with a number (see: figure 3). As opposed to 1.24% of all of the passwords containing a capital letter and a symbol. Of those passwords 29.11% start with a capital letter and end with a symbol (see: figure 4).

The difference between both patterns is 28.42 percentage points (σ=16,61).

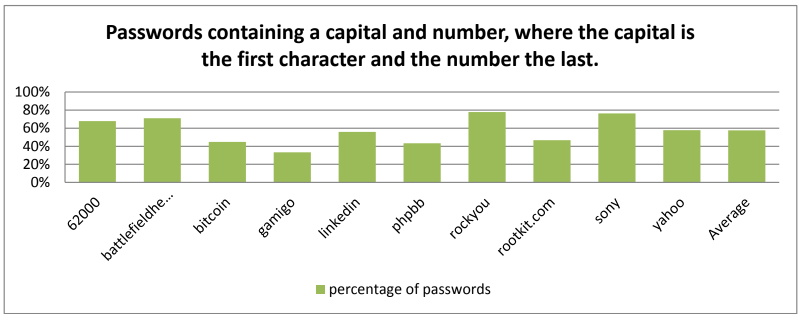


Figure 3: Experiment 3, capital first, number last

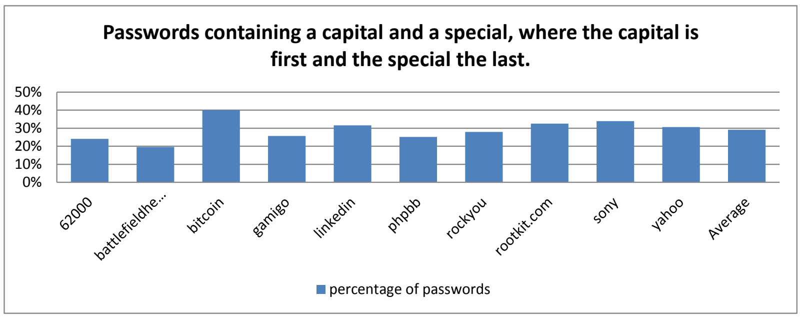


Figure 4: Experiment 3, capital first, special last

The results of all of our experiments seem to support Weir's statement. Yet it is interesting to note the differences in significance between the experiments. The differences measured in *experiment 1* are much more noticeable than the differences measured in *experiment 2*. An explanation for this could be that people use more than one number at the end of their passwords. We would like to see how these numbers compare to the situation when all the character positions of the passwords are analyzed, to be able to correctly and completely validate Weir's statement, yet this is beyond the scope of this research.

To make this research applicable in our password strength analysis we have chosen to generate even more specific patterns. This is why we have generated a list of patterns that can be directly inputted in the Hashcat tool, these patterns are named masks. Hashcat's masks abstract a password into its corresponding -per position- character set. A character set is denoted as a question mark followed by a letter defining one of four character sets e.g. 'P@ssw0rd' results in '?u?s?l?l?l?d?l?l'. Table 1 shows the complete definition of the password abstractions used by Hashcat which are denoted as masks.

Name	Abstraction	Character set
Lowercase	?l	abcdefghijklmnopqrstuvwxyz
Uppercase	?u	ABCDEFGHIJKLMNOPQRSTU-VWXYZ
Digit	?d	0123456789
Special	?s	!”#$%&’()*+,-./:;⇔@[\]^_‘{|}~

Table 1: Definition of Hashcat masks [28]

For every dataset we have generated a list of Hashcat masks that occurred a minimum of one percent, checked which masks were present in all of the datasets and calculated how much each mask occurs on average in each dataset. The outcome of this analysis can be found in table 2, it includes the average percentage of occurrence of each mask between all of the datasets analyzed and the standard deviation between the datasets as explained previously.

Hashcat mask	Percentage	σ
?l?l?l?l?l?l	9.87	4.6
?l?l?l?l?l?l?l?l	7.71	2.35
?l?l?l?l?l?l?l	6.76	2.7
?d?d?d?d?d?d	4.40	2.74
?l?l?l?l?l?l?d?d	3.70	1.6
?l?l?l?l?l?l?l?l?l	3.48	0.77
?l?l?l?l?l?d?d	1.72	0.59

Table 2: Masks present in every dataset

There are three important things to note from this table: **First** none of the seven masks present in all the datasets contains a '?s' indicating the lack of special characters. **Second** the masks can be separated into three simpler groups of passwords: passwords consisting of seven to nine lowercase letters, passwords consisting of six digits and passwords consisting of five or six lowercase letters followed by two digits. These are all very simple structures but are apparently used a lot. **Third** the differences in patterns between datasets can be very large, for example when we look at the '?l?l?l?l?l?l' mask the differences can be as large as 14.16%. Table 3 shows the differences for the '?l?l?l?l?l?l' mask broken down by dataset.

When we sum up the average percentages of the masks present in all of the datasets we get a total average of 37.64%. So when rounded for the sake of simplicity: on average almost 40% of all the passwords can be cracked using these seven password patterns. The remaining question to

Dataset	Percentage
62000	14.22
battlefieldheroes	8.45
bitcoin	6.54
gamigo	2.00
linkedin	3.25
phpbb	14.74
rockyou	12.23
rootkit.com	11.94
sony	16.16
yahoo	9.19

Table 3: Six lowercase letter mask presence per dataset

be answered then is: how long would it take to crack the passwords? To answer this question we first need to know how much different passwords will have to be tried or brute forced. Therefore we have calculated the key space associated with these masks:

$$(26^8)+(26^6)+(26^7)+(26^6*10^2)+(26^9)+(10^6)+(26^5*10^2)=5,678,752,184,704$$

A total of more than five and a halve trillion different possible passwords to be tried.

Dividing the number of possible passwords by the number of passwords our cracking setup is able to try each second [44] this results in a time estimate:

$$\left(\frac{\left(\frac{5,678,752,184,704}{13,000,000,000}\right)}{60}\right)=7.28\,minutes$$

In summary for all the datasets together –consisting of a total of 45 million passwords– it would take about 7 minutes to crack almost 40% of the passwords by using exhaustive search or 'stupid' brute force methods.

Another way to look at this data is to weigh the highest percentage of total occurrence as the most important factor instead of weighing the amount of datasets where the mask occurs in as the largest factor. In our previous analysis it is possible for a mask to occur in nine datasets with an average percentage of 50% yet that specific mask would not appear in our results because one dataset lacks this pattern.

Yet when we do this a lot of the masks are only present in one dataset. Of the forty-nine most occurring masks twenty-one only occur in one dataset: seventeen masks from Gamigo, two masks from RockYou, one mask from Battlefield Heroes Beta and one mask from phpBB [45]. Because of the large number of masks only present in the 'Gamigo' dataset – which as earlier noted contains some strange patterns see appendix B [43] – we require masks to occur in more than one dataset. This resulted in a total of twenty-eight masks; these masks account on average for 72.1% of all of the password in the datasets. But it they would take more than 88 days to brute force [46]. By removing the three masks that take the most time to brute force the percentage of cracked passwords is reduced by just 5% but the cracking time reduces to less than an hour. Appendix P [47] contains these masks and furthermore: the average percentage per dataset, the standard deviation amongst the datasets, the cumulative percentage of the masks, the key space of each mask and the time it would take to exhaust the whole keyspace of each mask (using the benchmark of thirteen billion MD5 hashes a second [44]). Please note the standard deviation in the table, once again suggesting that there is a significant spread between the percentage of patterns occurring in each dataset.

This analysis has learned us three different things about the patterns inside passwords: **First** a large portion of the passwords in the datasets can be abstracted into simple patterns. **Second** we know what those patterns are. **Third** we know what the average occurrence rate of each individual pattern is. The question remaining: In what order should each pattern be filled out? For instance, the most significant pattern is a password consisting of six lowercase letters, instead of guessing "aaaaaa" the password "monkey" is much more likely to occur and should therefore be guessed earlier on in the cracking process. One solution to this problem is to analyze in what order the characters are distributed in our large sample of passwords. This letter frequency analysis results in the occurrence rate of each character per position.

What is the average probability of a character occurring on a position in a password?

Because our datasets still contain some bogus passwords of unrealistic length, we limited our question to the first ten positions and to an occurence of more than one percent per position.

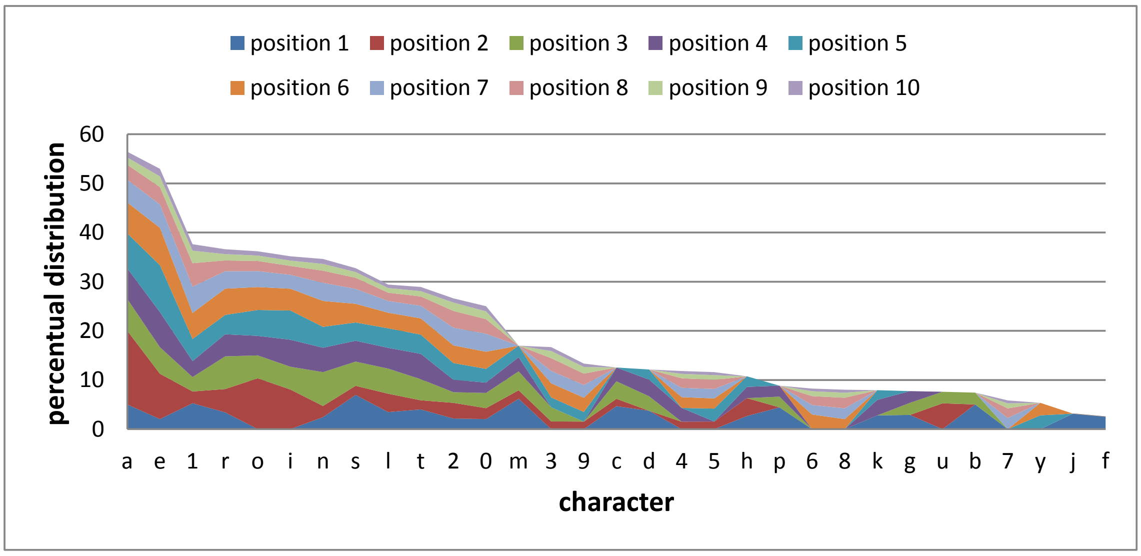


Figure 5: Per position letter frequency of all the datasets

Figure 5 shows that most vowels (a, e, i, o, u) are ranked very high, leading us to believe that the distribution of characters in password is similar to that of natural languages.

We compared the distribution of our password list to that of English text [29]. Because no English words contain numbers, we have removed them from the comparison. Table 4 shows a comparison between the distribution of letters in the password datasets and that of English text per character ordered by their frequency. It is interesting to note the 'oins' pattern that is exactly the same.

The figures in Appendix T [51] show a more detailed comparison between the letter frequency of passwords and that of English text, including the percentages of occurrence per position.

Passwords	a	e	r	o	i	n	s	l	t	m	c	d	h
English	e	t	a	o	i	n	s	r	h	l	d	c	u

Table 4: Letter frequency of passwords compared to English text

This letter frequency analysis can be further abstracted into a probabilistic process or Markov Model. Markov Models are used in speech and text recognition systems. When calculating the global letter frequency (not per position as we did) this equals a *Zero Order Markov Model*, it is a Markov Model without any memory, the model has no knowledge of the probability that a particular letter is preceded or followed by other letters. A *First Order Markov Model* has knowledge of the current letter and selects the next letter based on that. For example the letter 't' in English text is more likely to be followed by a 'h' than by a 'q', using a Markov Model trained on English text the word 'the' will be constructed far earlier than the gibberish word 'tqe'. The letter frequency analysis has shown that this is no different with passwords. Additionally any higher order Markov Model will take more of the preceding or following characters into account. A similar concept is already implemented in several password cracking tools. The Hashcat tool uses a per position Markov optimization, it considers both the position and the character that came before it.

Engineering - Password Cracking Setup

In this section we compare the hardware and tools of previously built password cracking systems, we describe the system we have built for the purpose of password cracking and compare the default implementation of the Hashcat tool to the patterns we found in our earlier research.

Before building a system for the purpose of password cracking we scoped the problem by defining some restrictions: the system could only consist of publicly available hardware, should make use of GPGPU ('General-Purpose computing on Graphics Processing Units') so no special purpose developed hardware like ASICs (Application-specific integrated circuits) or FPGAs (Field-programmable gate arrays) [30]. Furthermore it should make use of readily available software or tools, as GPGPU programming and the cryptographic problems of hashes will both raise very specialistic problems. Lastly the cost of the system should not exceed €1500.

The comparison between existing systems and tools is a problem that is somewhat hard to quantify as each system consists of different versions of hardware and tools. Therefore the benchmark of each system contains different quantifiers in the speed of password cracking. The only clear indicator available in all of the benchmarks is the speed of the MD5 hashing algorithm. We have compiled a table of three earlier purpose built password cracking systems and included the system we built, which can be found in appendix Q [48]. This table includes each system's: software, hardware, price, build date, photo and remarks. The power consumption is stated for the systems of which those figures are available.

An interesting figure to note from appendix Q is the sole use of AMD video cards where one would expect NVIDIA video cards to also be used. The reason that AMD cards are used much more than NVIDA video cards is because the AMD video cards are about five times faster when cracking passwords. We did not expect this as the software platforms used for GPGPU would suggest otherwise. When harnessing the power of GPGPU there are two competitive platforms: OpenCL and CUDA. CUDA is a proprietary platform developed by NVIDIA whereas OpenCL is an open platform originally authored by Apple and now developed by the Khronos Group [31]. Due to its proprietary background CUDA software can only run on NVIDIA hardware but OpenCL software can run on all different kinds of hardware, it can therefore run both on NVIDIA, AMD and even Intel hardware. Because of this OpenCL software has to have a higher level of abstraction than CUDA software. Normally extra abstraction layers cause extra overhead and result in slower performance. However, because of our specific needs this is not the case with our application namely password cracking.

When cracking passwords the overall speed largely depends on the speed with which the cryptographic hashes can be calculated. Most hash types depend on similar instructions and the instruction set implemented in AMD hardware –both GPU and CPU– supports these specific instructions. Instead of having to execute three different instructions when using the NVIDIA instructions, AMD hardware can do these operations using only one instruction. Appendix R [49] contains an example of this; the pseudocode for the MD5 hashing algorithm, the highlighted lines are the instuctions optimised on AMD hardware.

Additionally the current AMD hardware has a much higher 'bang for buck' or performance to money ratio than NVIDIA's equally priced cards. AMD's hardware has more raw computing power; more stream processors (computing units) clocked at higher speeds:

GeForce GTX670 1344 cores * 915 MHz [32] vs. Radeon HD 7970 2048 cores * 925 MHz [33] [4]

Both the optimized instruction set and the larger 'bang for buck' ratio are the reason AMD video cards are used when trying to crack passwords. Therefore we will also use a AMD Radeon HD 7970 in our password cracking setup as this gives the best 'bang for buck' ratio at the moment.

A thing to keep in mind is the decreasing marginal returns when investing in hardware: *"A single desktop with four Radeon HD 6990s for $3000 will increase cracking speed by 160 times. Buy [sic] a second such system, for another $3000, will only double your cracking speed after that."* [34] Furthermore because the time it takes to crack a password grows exponentially with each –randomly selected– added character, a very expensive –and therefore much faster– password cracking system only help a little when applying 'stupid' brute force methods to crack a password. Figure 6 shows the difference between the earlier mentioned systems [48] which are priced in a range from $2,000 to $20,000.

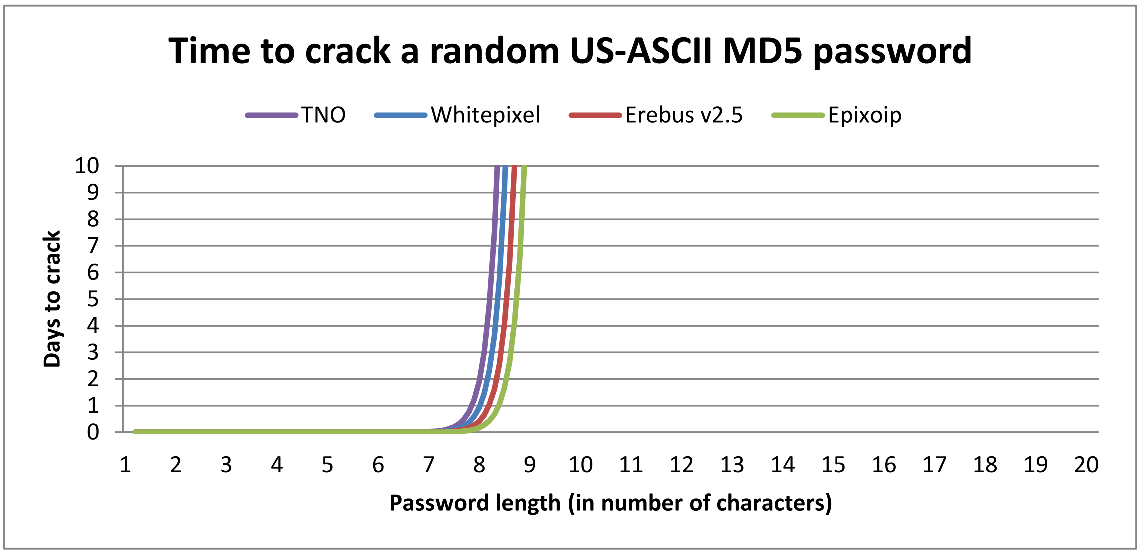


Figure 6: A system ten times more expensive cracks a truly random passwords only slightly faster.

Now that we know what hardware to use we want to know what different tools are available, what the differences are and which one is the most suitable for our purposes. The requirements are: active development, active community, cracking speed, number of hash types supported and of course GPGPU support. This narrows our possible tooling down to two competitors, each with its own benefits and drawbacks. *John the ripper*: Dating back to the 1990's John the Ripper is by far the most mature tool of the two, combined with a large community and immense hash type support

[4]both approximately the same price at the time of writing; €350

it is a very powerful tool. Most of the tool is written to run on the CPU. GPGPU support was added by the community and can be considered a beta version. It has the advantage of being open source therefore it is possible to extend the codebase. If a specific hash type is not supported anyone can write the code needed for its support. Therefore John the Ripper supports a lot of product specific hashing algorithms. *Hashcat*: Originally developed because other tools did not support parallelization. Hashcat evolved from a multithreaded CPU tool into a GPGPU tool. This results in high cracking speeds, it claims to be the world's fastest cracker in a multiple of hash types. Despite being free to use, unfortunately the tool is closed source, this might be a disadvantage when the support for a specific hash type is needed as this can only be requested and cannot be developed by ourself. However, the tool already has a multitude of hash types implemented; combined with the active development we consider this is a bearable risk. Furthermore Hashcat is available in three different versions: a version implemented on CPU, a version implemented on the GPU used to crack single hashes and a version implemented on the GPU used to crack a large list of hashes. The ability to crack large lists of hashes was interesting to us as it can help us to validate our earlier research. We have used both John the Ripper and Hashcat in our system yet have mainly used the Hashcat tool. Because of its more stable support of GPGPU, its very active development and community.

We have compared Hashcat's default settings to the patterns found in our research. Hashcat's default settings consist of two parameters: a mask and Markov based statistics. First we will discuss the default mask and secondly the Markov statistics.

Hashcat has an 'increment mode': instead of just brute forcing a mask of a specific length it will then iterate through that mask at given length. For example if one would enter the mask of six lowercase letters ('?l?l?l?l?l?l'), with 'increment mode' enabled, Hashcat will first brute force the length one mask ('?l') secondly the length two mask ('?l?l') etcetera until it has brute forced the complete length 1-6 lowercase letter key space. Furthermore Hashcat has the ability to define custom character sets, where we first had only the '?l', '?u', '?d' and '?s' character sets. Using custom character sets we can for example let the '?1' mask represent both the uppercase and lowercase letter character sets. Both these options are used by default when brute forcing a password. Table 5 shows the default Hashcat mask but because Hashcat has its increment mode enabled by default the default mask is broken into a separate mask for each length. The table in appendix S [50] shows the dissection of each mask brute force by Hashcat by default.

Denominator	Charset	Keyspace
?1	?l?d?u	62
?2	?l?d	36
?3	?l?d*!$@_	41
?1?2?2?2?2?2?2?3?3?3?3?d?d?d?d		

Table 5: Hashcat's default mask

It is interesting to note that Hashcat's default mask already implements all common masks and patterns we have found in our earlier research. Yet our masks are more specific as they reduce the key space even more and are therefore better when brute forcing slow hashes like bcrypt. It is our understanding that the default mask of Hashcat is already based on cracked passwords.

Secondly we compare Hashcat's default Markov statistics to statistics generated using the different datasets we have available. We have run one hour cracking sessions, each session based on Markov statistics from a different dataset. The dataset we set out to crack was the 'Battlefield Heroes Beta' dataset, this is the dataset with the least amount of previously cracked hashes. We have used the previously cracked hashes from the dataset to create Markov statistics and run a cracking session using those statistics. The results can be viewed in table 6.

Dataset	Cracked hashes
62000	19,505
battlefieldheroes	22,879
bitcoin	16,346
gamigo	20,442
linkedin	20,360
phpbb	20,381
rockyou	20,089
rootkit.com	19,852
sony	20,875
yahoo	21,032

Table 6: Results of a one hour long cracking session on the Battlefield Heroes Beta hashes. Trying to crack the entire length eight key space.

By default Hashcat is trained on the 'RockYou' dataset and does not update its Markov statistics dynamically. The table shows that using cracked hashes from the same set to generate Markov chains results in the faster cracking of hashes, a difference of cracked hashes of 12.2% between the default statistics and those generated from the pre-cracked hashes ('RockYou' 20,089 & 'Battlefield Heroes Beta' 22,879). The vast majority of the time a Markovian based attack will recover more passwords than a standard brute force based attack (see figure 7). Yet the difference will only be noticeable when the key space of the mask to be brute forced cannot be fully exhausted (as both attacks will crack all of the hashes in the end). That is why we tried to brute force the full length 8 key space, which would take more than 17 days and stopped the cracking session after one hour.

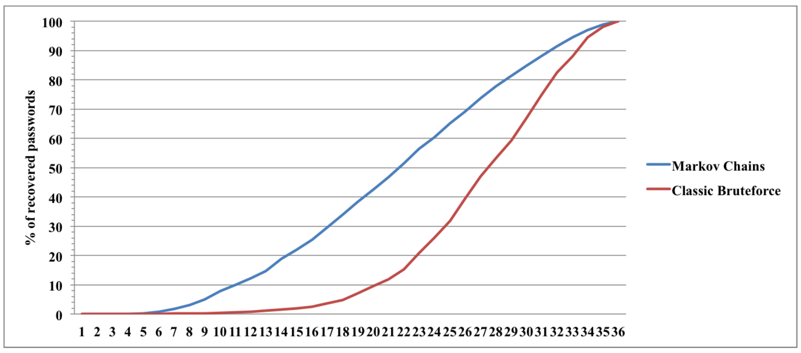


Figure 7: *"Percentage of recovered passwords (Rockyou) progressions for 6 character passwords. Comparison of Markov Chains Dictionary Attack Vs. Classic Brute-force at 36 pre-set time intervals"* [19]

Design - Password strength analysis demonstrator

Our password strength analysis demonstrator will assess the strength of a password based on automated password cracking techniques. It will be able to show the danger of statistical attacks which exploit the non-randomness of user-created passwords.

Earlier examples of more advanced password strength meters are known to us [20, 21]. These incorporate some detection of common patterns yet do not directly model password cracking attacks against the password to result in a figure of password strength. We have shaped our password strength analysis to incorporate attacks used by professional password crackers [22]. Which are available by default in password cracking tools [23]. Each of the steps we do to test the strength of a password can be viewed as a stage of a multistage rocket. Each consecutive stage will have to adhere to rule stated in the previous one.

We have an advantage over a normal password cracking as we already have the plaintext password available and consequently do not have to generate all possible password-guesses. Our algorithm is able to check if the plaintext password suffices to any of our rules and if it does it will be considered a crackable password and therefore weak. Instead of having to check all of the possible guesses. Following are the rules used to assess the strength of a password:

To 'survive' an automated password cracking attack a password may

Brute force attack [5]

- not consist of a single character set.
- be no smaller than twelve characters when using two character sets: numbers and lowercase or uppercase and specials etcetera.
- be no smaller than eleven characters when using three character sets: lowercase, numbers and specials or uppercase, lowercase and numbers etcetera.
- be no smaller than ten characters when using the whole ASCII charset: lowercase, uppercase, numbers and symbols.

Wordlist attack

- not be in any of our wordlists.
- not contain a large consecutive part of a word in any of our wordlists.

[5] We use the speed of 'stupid' brute force of the cracking systems stated in [48] as a lowerbound.

Rule-based attack

- be no simple permutation of any of the words in the wordlists
- not contain repetitions, sequences or combinations of those two.

Mask attack

- not consist of a easy to brute force mask. [6]

Markov attack

- not consist of common transitions between characters.

After the last step only passwords lacking common patterns –consequently hard to guess passwords– are left. We are not able to subvert all common patterns as these could originate a vast amount of sources like pop-culture or demographic properties.

We are aware that we assumed one of the worst case scenarios, an offline cracking attack of MD5 passwords. Yet this is not an unreasonable assumption as our datasets show. All of these sets were either leaked plaintext or leaked hashed with a 'fast' hashing function. This motivates our research as it could be a concrete danger for one's password to be part of a leak. We do not return any time estimation as this is very dependent to the hardware and hashing algorithm used which could again result in a 'false sense of security'. We only suggest if the password can be cracked by an automated password cracking attack.

One final note to make is that one might assume this to be the ideal password meter to accompany an account registration form. We would advise not to tread lightly on this subject, our password strength analysis might reduce the global key space of a service significantly when implemented directly as such a password meter.

[6]The top mask from our research cannot be used, as these are already excluded by the rules stated with the brute force attack. We use other masks which have a lower probability.

Conclusions

The goal of this thesis was to build a state-of-the-art password strength analysis demonstrator. We set forth to help users make their passwords more resistant against automated password cracking attacks by assessing the password strength of a password using a simulation of a password cracking attack.

The first thing we have done was to research the possible patterns inside passwords, which could lower the overall password strength. We have done this by analysing 45 million user-created passwords. We have run experiments to check if the assumtions made by other researchers were significant and interesting and have found that a large portion of the analyzed passwords consist of simple and easy to crack patterns. Furthermore we found that the letter frequency of the passwords is similar to that of natural languages and compared it with English text.

Second we have studied the hardware and tools used in existing password cracking systems and have assembled our own password cracking system which we have used to compare the default settings of the Hashcat tool to the patterns found in our earlier research. We note that the reason why AMD video cards are used for the purpose of password cracking has to do with the instuction set implemented in the hardware and the bigger performance to money ratio. Furthermore we found that the Hashcat tool has a strong implementation of default settings as it is already based upon the probablities and patterns of user-created passwords. We suggest some improvements upon the default settings and argue these could lead to sigificant improvements when cracking slow hashes.

During the aforementioned two steps we have studied the psychological and technical factors which together make up for the strength of a password. We have used the results of these steps to implement a system that assesses the strength of a user's password based upon a simulation of automated password cracking attack. We are conviced our system is able to give users better advice about the strength of their passwords because it makes a user aware of the common patterns contained in his or her password and therefore hopefully nudges a user to use a password that lacks these common patterns. We are aware that our system is not able to guarantee one's password strength yet it is a significant improvement over most of the password strength analysis algorithms available.

Recommendations

Our recommendations are fourfold, we have defined four groups we would like to give specific advice: end-users, developers, large technology companies (e.g. Google, Intel, Microsoft, denoted Fortune 500) and institutions (e.g. National Institute of Standards and Technology (NIST), The Internet Engineering Task Force (IETF)). We think more industry-wide collaboration leads to a more secure cyber-domain.

A success story of such collaboration is development and implementation of the 'Time-based One-time Password Algorithm' (TOTP) [36]. This algorithm originated from the Initiative for Open Authentication (OATH) [37] and is now implemented for the use of two factor authentication by i.a. Google, Microsoft and Dropbox. TOTP is implemented for use with a smartphone application (Android, IOS, Windows Phone) as the second factor in the authentication scheme.

An example of a collaboration to be put into a higher gear is the adoption of hash functions like bcrypt and scrypt. Developers seem reluctant to implement these hashing functions, possibly due to the requirement to support legacy systems running older versions of software which do not have native support for these hashing functions. On the other hand PHP, for example, supports bcrypt from the 5.3.0 release (June 30, 2009) onwards so a better explanation could be that developers do not implement these hashing functions due to the lower priority of password security and which, in turn, is the result of uninformed and unaware developers. We are eager to see what happens with the IETF draft of scrypt [35]. This should hopefully lead to the implementaion of scrypt in more systems and software.

End-user

- Use a password manager when possible [38, 39, 40]
- Do not use common password patterns
- Use lengthy passwords
- Use all ASCII character sets (lowercase, uppercase, numbers, specials)
- Use different passwords for different services
- Use two factor authentication when available (e.g. Gmail, Hotmail, Twitter, Facebook, Dropbox)

Developers

- Implement scrypt for the use of password hashing

- Use standardized cryptographic functions, do not try to create your own
- Implement two factor authentication (e.g. TOTP)

Fortune 500

- Support TOTP
- Give uniform advice to users and developers
- Encourage users and developers to adopt industry standards
- Inform developers with easy to understand code examples for different programming languages
- Inform users with tutorials and videos

Institutions

- Encourage the discussion of new industry standards (e.g. scrypt)

References

[1] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, Cormac Herley, and Washington Redmond. Does my password go up to eleven? the impact of password meters on password selection. 2013.

[2] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, et al. How does your password measure up? the effect of strength meters on password creation. In *Proc. USENIX Security*, 2012.

[3] Microsoft - Check your password is it strong?. (2013, June) [Online]. `https://www.microsoft.com/security/pc-security/password-checker.aspx`

[4] Intel - How Strong is Your Password?. (2013, June) [Online]. `https://www-ssl.intel.com/content/www/us/en/forms/passwordwin.html`

[5] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 162–175. ACM, 2010.

[6] Charles Matthew Weir. *Using probabilistic techniques to aid in password cracking attacks*. PhD thesis, 2010.

[7] Marcus Bakker and Roel Van Der Jagt. Gpu-based password cracking. 2010.

[8] Martijn Sprengers. *GPU-based Password Cracking*. PhD thesis, Master's thesis, Radboud University Nijmegen, 2011.

[9] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–597, 1979.

[10] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Advances in Cryptology–EUROCRYPT 2005*, pages 19–35. Springer, 2005.

[11] Gaëtan Leurent. Md4 is not one-way. In *Fast Software Encryption*, pages 412–428. Springer, 2008.

[12] Martin Hellman. A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401–406, 1980.

[13] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology-CRYPTO 2003*, pages 617–630. Springer, 2003.

[14] Hashcat. (2013, June) [Online]. http://hashcat.net/

[15] John the Ripper. (2013, June) [Online]. http://www.openwall.com/john/

[16] Niels Provos and David Mazieres. A future-adaptable password scheme. In *Proceedings of the Annual USENIX Technical Conference*, 1999.

[17] Colin Percival. Stronger key derivation via sequential memory-hard functions. In *BSDCan 2009: The Technical BSD Conference*, 2009.

[18] Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 523–537. IEEE, 2012.

[19] Chrysanthou Yiannis *Modern Password Cracking: A hands-on approach to creating an optimised and versatile attack* PhD thesis, Department of Mathematics, Royal Holloway, University of London, 2013.

[20] zxcvbn - realistic password strength estimation. (2013, June) [Online]. https://tech.dropbox.com/2012/04/zxcvbn-realistic-password-strength-estimation/

[21] How Secure Is My Password?. (2013, June) [Online]. https://howsecureismypassword.net/

[22] Anatomy of a hack: How crackers ransack passwords like q̈eadzcwrsfxv1331∵ (2013, June) [Online]. http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/

[23] Hashcat - Attack modes. (2013, June) [Online]. http://hashcat.net/wiki/#attack_modes

[24] phpBB: Downtime and Server Compromise. (2013, June) [Online]. http://area51.phpbb.com/phpBB/viewtopic.php?f=3&t=29973

[25] An Update on LinkedIn Member Passwords Compromised. (2013, June) [Online]. http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised/

[26] PACK. (2013, June) [Online]. http://thesprawl.org/projects/pack/

[27] MySQL Case Sensitivity in String Searches. (2013, June) [Online]. http://dev.mysql.com/doc/refman/5.0/en/case-sensitivity.html

[28] Hashcat Mask Attack. (2013, June) [Online]. http://hashcat.net/wiki/doku.php?id=mask_attack

[29] English Letter Frequency Counts: Mayzner Revisited. (2013, June) [Online]. http://norvig.com/mayzner.html

[30] What is the Difference Between a FPGA and an ASIC?. (2013, June) [Online]. http://www.xilinx.com/fpga/asic.htm

[31] The open standard for parallel programming of heterogeneous systems. (2013, June) [Online]. http://www.khronos.org/opencl/

[32] NVIDIA GeForce GTX 670 Specifications. (2013, June) [Online]. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-670/specifications

[33] AMD Radeon HD 7970 Specifications. (2013, June) [Online]. http://www.amd.com/us/products/desktop/graphics/7000/7970/Pages/radeon-7970.aspx#3

[34] Password cracking, mining, and GPUs. (2013, June) [Online]. http://erratasec.blogspot.nl/2011/06/password-cracking-mining-and-gpus.html

[35] The scrypt Password-Based Key Derivation Function draft. (2013, June) [Online]. http://tools.ietf.org/html/draft-josefsson-scrypt-kdf-00

[36] TOTP: Time-Based One-Time Password Algorithm RFC. (2013, June) [Online]. http://tools.ietf.org/html/rfc6238

[37] OATH - initiative for open authentication. (2013, June) [Online]. http://www.openauthentication.org/

[38] KeePass Password Safe. (2013, June) [Online]. http://keepass.info/

[39] LastPass - Password Manager, Form Filler, Password Management. (2013, June) [Online]. https://lastpass.com/

[40] 1Password. (2013, June) [Online]. https://agilebits.com/onepassword

[41] Parmy Olson. *We are anonymous: Inside the hacker world of LulzSec, Anonymous, and the global cyber insurgency.* Little, Brown, 2012.

[42] Nico van Heijningen: Apendix A - dictclean.php

[43] Nico van Heijningen: Appendix B - Dataset remarks and numerals

[44] Nico van Heijningen: Appendix M - Cracking System Benchmarks

[45] Nico van Heijningen: Appendix N - Masks only occurring in one dataset

[46] Nico van Heijningen: Appendix O - Masks occurring in more than one dataset

[47] Nico van Heijningen: Appendix P - Masks occurring in more than one dataset - without three largest masks

[48] Nico van Heijningen: Appendix Q - Comparison of systems

[49] Nico van Heijningen: Appendix R - MD5 pseudocode

[50] Nico van Heijningen: Appendix S - Hashcat's default bruteforce mask increment mode

[51] Nico van Heijningen: Appendix T - Per position letter frequency heatmaps

Evaluation

Looking back at the writing of this thesis and the graduation process as a whole I can say that it was a very fulfilling exercise. Information theory, the theory of probability, cryptography and statistics were all needed to be able to complete this thesis. These subjects were all taught at school yet all are a very limited part of the curriculum of the CMI-Program Technical Informatics at the Rotterdam University. So some self-dependence was needed to bring my knowledge of these subject to a higher level. I have learned to apply these subjects to real problems and questions; to put the theory into practice.

The subject of the thesis surrounding passwords is one that I very found interesting one that is active and contains a lot of recent developments. Because of the state-of-the-art requirement of the thesis I had to delve into the literature and analyze related or existing work. This helped me to create my own image of possible solutions to the problems along the way.

This assignment was a combination of research and engineering including both psychological and technical factors. Although I liked this very much having touched all facets of the assignment this meant the size of the subject increased by a multitude not expected at the beginning of this thesis. Through deliberate scoping I still had the opportunity to specialize on parts I thought were of interest yet still resulting in the final design and realization of a product which now incorporates these parts.

Furthermore the contact with colleagues was a very satisfying part of this thesis. The presentations and demonstrations I gave to colleagues at TNO have helped me to get my story straight and I think it helped them in with some knowledge and new information. The brainstorming with and input of: researchers, developers, enthusiasts and less technical people have helped shape this thesis into what it has become today. This is part of the reason I am going to attend several conferences on the subject of authentication and computer security in the United States of America.

Finally I have done recommendations to multiple parties instead of just one as I think this is an industry-wide problem which cannot be solved by a single party. The solutions I have put forward are not necessarily the final solutions to this problem but are a viable solution in the best of my belief.

Appendix A

dictclean.php

```
<?php
define('DICTCLEAN_VERSION', '0.1');

#
# 1. Options
#
$options = getopt('', array(
  'help',
  'list-encodings',
  'encoding:',
  'dictfile:',
  'cleanfile:',
  'dirtyfile:',
));

# --help
if (isset($options['help'])) {
  echo 'dictclean ', DICTCLEAN_VERSION, ", T. Alexander Lystad <
    tal@lystadonline.no> (www.thepasswordproject.com)

Usage on Windows: php -f dictclean.php -- [switches]
Usage on Linux: ./dictclean.php -- [switches]

Example use on Windows: php -f dictclean.php -- --dictfile rockyou.txt --
    cleanfile rockyou.clean.txt
Example use on Linux: ./dictclean.php -- --dictfile rockyou.txt --cleanfile
    rockyou.clean.txt

Switches:
--help \t\t\t Show help
--list-encodings \t List available encodings
--encoding \t\t The encoding you want to check for. Must be listed in --list-
    encodings. Defaults to UTF-8. Example: --encoding ISO-8859-1
--dictfile \t\t The file to analyze. Example: --dictfile dictfile.txt
--cleanfile \t\t Generate cleaned up dictfile. All lines from dictfile with
    valid encoding will be written to this file. Example: --cleanfile cleandict
    .txt
--dirtyfile \t\t Generate dirty dictfile. All lines from dictfile with invalid
    encoding will be written to this file. Example: --dirtyfile dirtydict.txt";
  exit;
}
```

```
# --list-encodings
if (isset($options['list-encodings'])) {
  echo 'Available encodings on your system: ', "\n", implode("\n",
    mb_list_encodings());
  exit;
}

# --encoding
if (isset($options['encoding'])) {
  define('WANTED_ENCODING', 'UTF-8');
}
if (!defined('WANTED_ENCODING')) {
  define('WANTED_ENCODING', 'UTF-8');
}

# --dictfile
if (isset($options['dictfile'])) {
  define('DICTIONARY_FILE', $options['dictfile']);
}
if (!defined('DICTIONARY_FILE')) {
  echo 'You have to specify the file you want to analyze. Example: --dictfile
    dictionary.txt';
  exit;
}
if (!is_readable(DICTIONARY_FILE)) {
  echo 'Could not read file \'', DICTIONARY_FILE, '\'. Please specify a correct
     path for the file you want to analyze.';
}

# --cleanfile
if (isset($options['cleanfile'])) {
  $cleanHandle = fopen($options['cleanfile'], 'w');
}

if (isset($options['dirtyfile'])) {
  $dirtyHandle = fopen($options['dirtyfile'], 'w');
}


#
# 2. Meat
#
echo 'dictclean ', DICTCLEAN_VERSION, ' report (www.thepasswordproject.com)', "
   \n\n";
$invalidCount = 0;
$lineCount = 1;
$inHandle = fopen(DICTIONARY_FILE, 'r');
while (($line = fgets($inHandle)) !== false) {
  if (mb_check_encoding($line, WANTED_ENCODING)) {
    if (isset($cleanHandle)) {
      fwrite($cleanHandle, $line);
    }
  } else {
    if (isset($dirtyHandle)) {
      fwrite($dirtyHandle, $line);
    }
    $detectedEncoding = mb_detect_encoding($line, null, true);
    if ($detectedEncoding) {
      $detectedString = $detectedEncoding.' encoding was detected';
    } else {
      $detectedString = 'Encoding could not be detected';
    }
    echo 'Invalid ', WANTED_ENCODING, ' at line ', $lineCount, ': \'', trim(
    $line), '\' (', $detectedString, ')', "\n";
    $invalidCount++;
  }
  $lineCount++;
}
echo 'Lines with invalid ', WANTED_ENCODING, ': ', $invalidCount, '/',
    $lineCount, ' (', round(($invalidCount/$lineCount)*100, 4), ' %)';
```

bijlagen/dictclean.php

Appendix B

Dataset remarks and numerals

62000 random logins - Lulzsec, it is unknown where these passwords originated from and how old they are [1].

Battlefield heroes beta, a free cartoon-styled video game published by EA.

Mt. Gox, one of the largest bitcoin exchanges. What makes leak interesting is that these passwords are the equivalent of banking passwords and the target group is very technically savvy. Out of the total 61,000 passwords leaked only the 1,764 MD5 hashes have been tried to crack. Because the remaining 59,236 stronger MD5(unix) hashes are much harder to crack.

Gamigo, a free online social gaming site. The leak contains a lot of ?l?l?l?u?u?u?d?d?d masks for some reason, about 3% of all passwords F.

Linkedin, a social networking website. The leak contains 3,521,180 hashes starting with 00000, and 2,936,840 unmasked hashes. The hashes starting with zeroes were probably a parsing error by the attacker.

phpBB, open source internet forum software. The full leak contains ˜400,000 hashes.

Rockyou, a free online social gaming developer. By far the largest leak. Rockyou's own password policy only required a five-character password, and did not permit special characters. But since many of Rockyou's applications were able to crosspost across multiple social network sites, the password list contains multiple passwords from the same users originating from multiple social networking sites like MySpace, Facebook, etc. Also, there is some 'junk' in there that probably represents the attacker parsing the SQL dump incorrectly. Finally, since this represents multiple site passwords, there is no overall password creation policy applied to these passwords.

Rootkit.com, a popular website for discussion and analysis of rootkits. This hack was part of the attack by 'Anonymous' (most of which became members of Lulzsec [41]) on the technology security company HBGary. It includes several passwords from MITRE employees.

Sony pictures, from sonypictures.com the main website for Sony's film and television franchise. The leak contains data related to Sony sweepstakes involving AutoTrader.com, "Summer of Restless Beauty" and "Seinfeld - We're Going to Del Boca Vista!". The Autotrader.com dataset only contains records of elderly people (born before 1944) [`https://twitter.com/LulzSec/status/77845443383525377`], We didn't use it as the data is corrupted and contains a lot of duplicates. The Beauty & Seinfeld datasets are used.

[1]The release noted: "these are random assortments from a collection, so don't ask which site they're from or how old they are, because we have no idea. We also can't confirm what percentage still work, but be creative or something."

Yahoo! Voices, a voice over IP service provided by Yahoo! and hacked by the hacking group D33ds Company.

Name of leak	Release date	Hashing function	# leaked passwords	% list used	% unique
62000 random logins - Lulzsec	June 16, 2011	None	62,156	99.89	76.93
50 Days of Lulz - Battlefield Heroes Beta	June 25, 2011	MD5	548,773	50.55	98.86
Bitcoin	June 19, 2011	MD5	1,764	69.33	89.94
Gamigo	July 6, 2012	MD5	7,004,341	90.03	99.98
Linkedin	June 5, 2012	SHA1	6,458,020	83.67	91.20
phpBB	January 31, 2009	MD5	259,424	98.81	72.24
RockYou	December 15, 2009	None	32,603,388	100.00	43.99
Rootkit.com	February 6, 2011	MD5	71,228	93.81	81.44
Sony Pictures	June 2, 2011	None	36,310	99.99	80.92
Yahoo	July 12, 2012	None	442,838	100.00	77.35

Table B.1: Dataset numerals

Appendix C

Pack - 62000

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
   faster parsing.
[*] Analyzing passwords: ../../lists/62000/passwords/sorted.txt
[+] Analyzing 100% (62086/62086) passwords
    NOTE: Statistics below is relative to the number of analyzed
         passwords, not total number of passwords

[*] Line Count Statistics...
[+]                           6: 31% (19296)
[+]                           8: 25% (16135)
[+]                           7: 16% (10525)
[+]                           9: 09% (6199)
[+]                          10: 06% (4065)
[+]                          12: 03% (2441)
[+]                          11: 02% (1751)
[+]                           4: 01% (672)

[*] Mask statistics...
[+]                   allstring: 43% (27092)
[+]                 stringdigit: 26% (16636)
[+]                    alldigit: 19% (12164)
[+]                 digitstring: 03% (1977)
[+]                   othermask: 02% (1795)
[+]            stringdigitstring: 02% (1575)
[+]             digitstringdigit: 00% (427)
[+]           stringspecialdigit: 00% (159)
[+]          stringspecialstring: 00% (141)
[+]                stringspecial: 00% (89)
[+]         specialstringspecial: 00% (15)
[+]                specialstring: 00% (12)
[+]                   allspecial: 00% (4)

[*] Charset statistics...
[+]                   loweralpha: 43% (26707)
[+]            loweralphanum: 33% (21088)
[+]                  numeric: 19% (12164)
[+]     loweralphaspecialnum: 00% (615)
[+]            mixedalphanum: 00% (562)
[+]        loweralphaspecial: 00% (247)
[+]            upperalphanum: 00% (206)
[+]               mixedalpha: 00% (198)
[+]               upperalpha: 00% (187)
[+]     mixedalphaspecialnum: 00% (58)
[+]        mixedalphaspecial: 00% (20)
[+]     upperalphaspecialnum: 00% (19)
[+]        upperalphaspecial: 00% (11)
[+]                  special: 00% (4)

[*] Advanced Mask statistics...
[+]             ?l?l?l?l?l?l: 14% (8831)
[+]             ?d?d?d?d?d?d: 10% (6296)
[+]           ?l?l?l?l?l?l?l: 09% (5931)
[+]         ?l?l?l?l?l?l?l?l: 08% (5365)
[+]         ?d?d?d?d?d?d?d?d: 05% (3678)
[+]       ?l?l?l?l?l?l?l?l?l: 04% (2636)
[+]         ?l?l?l?l?l?l?d?d: 03% (1893)
[+]     ?l?l?l?l?l?l?l?l?l?l: 02% (1522)
[+]             ?l?l?l?l?l?d: 01% (1226)
[+]           ?l?l?l?l?l?d?d: 01% (1112)
[+]         ?l?l?l?l?d?d?d?d: 01% (1075)
[+]             ?l?l?l?l?d?d: 01% (976)
[+] ?l?l?l?l?l?l?l?l?l?l?l?l: 01% (971)
[+]         ?l?l?l?l?l?l?l?d: 01% (957)
[+]           ?l?l?l?d?d?d?d: 01% (804)
[+]   ?l?l?l?l?l?l?l?l?l?l?l: 01% (760)
[+]           ?l?l?l?l?l?l?d: 01% (710)
[+]       ?l?l?l?l?l?l?l?d?d: 01% (696)
[+]       ?l?l?l?l?l?d?d?d?d: 01% (646)

[*] Saving Mask statistics to ../../analysis/pack/masks/62000.
   csv
```

Appendix D

Pack - Battlefield Heroes Beta

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
   faster parsing.
[*] Analyzing passwords: ../../lists/battlefieldheroes/passwords
   /sorted.txt
[+] Analyzing 100% (277418/277418) passwords
    NOTE: Statistics below is relative to the number of analyzed
         passwords, not total number of passwords

[*] Line Count Statistics...
[+]                          8: 43% (120715)
[+]                          6: 23% (66170)
[+]                          7: 21% (58441)
[+]                          9: 09% (27203)

[*] Mask statistics...
[+]                stringdigit: 41% (115182)
[+]                  allstring: 29% (81044)
[+]                   alldigit: 11% (31497)
[+]                  othermask: 06% (17595)
[+]          stringdigitstring: 05% (16391)
[+]                digitstring: 03% (10285)
[+]           digitstringdigit: 01% (3594)
[+]         stringspecialdigit: 00% (840)
[+]        stringspecialstring: 00% (532)
[+]              stringspecial: 00% (315)
[+]              specialstring: 00% (78)
[+]       specialstringspecial: 00% (53)
[+]                 allspecial: 00% (12)

[*] Charset statistics...
[+]              loweralphanum: 51% (142399)
[+]                 loweralpha: 27% (74998)
[+]                    numeric: 11% (31497)
[+]              mixedalphanum: 06% (16846)
[+]              mixedalpha: 01% (5019)
[+]           upperalphanum: 00% (2011)
[+]    loweralphaspecialnum: 00% (1830)
[+]              upperalpha: 00% (1027)
[+]       loweralphaspecial: 00% (787)
[+]    mixedalphaspecialnum: 00% (661)
[+]       mixedalphaspecial: 00% (222)
[+]    upperalphaspecialnum: 00% (90)
[+]       upperalphaspecial: 00% (19)
[+]                 special: 00% (12)

[*] Advanced Mask statistics...
[+]        ?l?l?l?l?l?l?l?l: 08% (24862)
[+]            ?l?l?l?l?l?l: 08% (23451)
[+]        ?l?l?l?l?l?l?d?d: 07% (20675)
[+]          ?l?l?l?l?l?l?l: 06% (18451)
[+]            ?d?d?d?d?d?d: 05% (14040)
[+]        ?d?d?d?d?d?d?d?d: 03% (10760)
[+]        ?l?l?l?l?l?l?l?d: 03% (8949)
[+]        ?l?l?l?l?l?d?d?d: 02% (8191)
[+]        ?l?l?l?l?d?d?d?d: 02% (8120)
[+]          ?l?l?l?l?l?d?d: 02% (7882)
[+]      ?l?l?l?l?l?l?l?l?l: 02% (7023)
[+]            ?l?l?l?l?d?d: 02% (5812)
[+]          ?l?l?l?l?l?l?d: 02% (5605)
[+]          ?d?d?d?d?d?d?d: 01% (5433)
[+]            ?l?l?l?l?l?d: 01% (5081)
[+]      ?l?l?l?l?l?l?d?d?d: 01% (5015)
[+]      ?l?l?l?l?l?l?l?d?d: 01% (4245)
[+]          ?l?l?l?l?d?d?d: 01% (3328)
[+]            ?l?l?l?d?d?d: 01% (2969)

[*] Saving Mask statistics to ../../analysis/pack/masks/
    battlefieldheroes.csv
```

Appendix E

Pack - Bitcoin

```
[?] Psyco is not available. Install Psyco on 32−bit systems for
    faster parsing.
[∗] Analyzing passwords: ../../lists/bitcoin/passwords/sorted.
    txt
[+] Analyzing 100% (1223/1223) passwords
     NOTE: Statistics below is relative to the number of analyzed
          passwords, not total number of passwords

[∗] Line Count Statistics...
[+]                           8: 32% (400)
[+]                           6: 15% (191)
[+]                           9: 14% (172)
[+]                           7: 13% (161)
[+]                          10: 09% (113)
[+]                          12: 04% (57)
[+]                          11: 03% (48)
[+]                           5: 02% (34)
[+]                          14: 01% (17)
[+]                          13: 01% (16)

[∗] Mask statistics...
[+]                 stringdigit: 32% (393)
[+]                   allstring: 30% (372)
[+]                   othermask: 14% (176)
[+]           stringdigitstring: 09% (113)
[+]                    alldigit: 06% (77)
[+]                 digitstring: 03% (39)
[+]            digitstringdigit: 01% (20)
[+]               stringspecial: 00% (11)
[+]          stringspecialdigit: 00% (10)
[+]         stringspecialstring: 00% (8)
[+]               specialstring: 00% (3)
[+]        specialstringspecial: 00% (1)
```

```
[*] Charset statistics...
[+]              loweralphanum: 45% (557)
[+]                 loweralpha: 28% (348)
[+]              mixedalphanum: 10% (123)
[+]                    numeric: 06% (77)
[+]       mixedalphaspecialnum: 02% (35)
[+]       loweralphaspecialnum: 02% (27)
[+]          loweralphaspecial: 01% (20)
[+]                 mixedalpha: 01% (20)
[+]              upperalphanum: 00% (7)
[+]          mixedalphaspecial: 00% (5)
[+]                 upperalpha: 00% (4)

[*] Advanced Mask statistics...
[+]           ?l?l?l?l?l?l?l?l: 07% (92)
[+]               ?l?l?l?l?l?l: 06% (80)
[+]             ?l?l?l?l?l?l?l: 04% (57)
[+]           ?l?l?l?l?l?l?d?d: 03% (48)
[+]         ?l?l?l?l?l?l?l?l?l: 03% (38)
[+]           ?l?l?l?l?l?d?d?d: 02% (26)
[+]                 ?l?l?l?l?l: 01% (24)
[+]               ?d?d?d?d?d?d: 01% (22)
[+]             ?d?d?d?d?d?d?d: 01% (21)
[+]           ?l?l?l?l?d?d?d?d: 01% (18)
[+]           ?l?l?l?l?l?l?l?d: 01% (18)
[+]       ?l?l?l?l?l?l?l?l?l?l: 01% (18)
[+]               ?l?l?l?d?d?d: 01% (18)
[+]   ?l?l?l?l?l?l?l?l?l?l?l?l: 01% (17)
[+]           ?d?d?d?d?d?d?d?d: 01% (17)
[+]             ?l?l?l?l?l?d?d: 01% (13)
[+]             ?l?l?l?l?l?l?d: 01% (13)
[+]         ?l?l?l?l?l?l?l?l?d: 01% (13)

[*] Saving Mask statistics to ../../analysis/pack/masks/bitcoin.
    csv
```

Appendix F

Pack - Gamigo

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
    faster parsing.
[*] Analyzing passwords: ../../lists/gamigo/passwords/sorted.txt
[+] Analyzing 100% (6305237/6305237) passwords
    NOTE: Statistics below is relative to the number of analyzed
          passwords, not total number of passwords

[*] Line Count Statistics...
[+]                          10: 44% (2797830)
[+]                           8: 20% (1318414)
[+]                           9: 12% (768868)
[+]                           7: 06% (395013)
[+]                           6: 06% (381958)
[+]                          11: 04% (252719)
[+]                          12: 02% (180339)
[+]                          13: 01% (73383)

[*] Mask statistics...
[+]                  othermask: 31% (1985176)
[+]                stringdigit: 29% (1849460)
[+]                  allstring: 15% (1007788)
[+]          stringdigitstring: 13% (827442)
[+]                   alldigit: 04% (296965)
[+]                digitstring: 04% (255288)
[+]           digitstringdigit: 01% (70993)
[+]         stringspecialdigit: 00% (4807)
[+]        stringspecialstring: 00% (4489)
[+]              stringspecial: 00% (2052)
[+]       specialstringspecial: 00% (423)
[+]              specialstring: 00% (338)
[+]                 allspecial: 00% (16)

[*] Charset statistics...
[+]             loweralphanum: 65% (4119486)
[+]          mixedalphanum: 12% (813161)
[+]             loweralpha: 12% (788700)
[+]                numeric: 04% (296965)
[+]             mixedalpha: 03% (197963)
[+]          upperalphanum: 00% (47158)
[+]             upperalpha: 00% (21125)
[+]  loweralphaspecialnum: 00% (9630)
[+]     loweralphaspecial: 00% (5812)
[+]  mixedalphaspecialnum: 00% (2881)
[+]     mixedalphaspecial: 00% (1726)
[+]  upperalphaspecialnum: 00% (351)
[+]     upperalphaspecial: 00% (263)
[+]               special: 00% (16)

[*] Advanced Mask statistics...
[+]    ?1?1?1?u?u?u?d?d?d: 03% (191346)
[+]      ?1?1?1?1?1?1?1?1: 02% (160777)
[+]      ?1?1?1?1?1?1?d?d: 02% (146972)
[+]          ?1?1?1?1?1?1: 02% (126403)
[+]        ?1?1?1?1?1?1?1: 01% (115117)
[+]    ?1?1?1?1?1?1?1?1?1: 01% (114390)
[+]  ?1?1?1?1?1?1?1?1?1?1: 01% (106022)
[+]      ?d?d?d?d?d?d?d?d: 01% (92130)
[+]    ?1?1?1?1?1?1?1?d?d: 01% (84706)
[+]  ?1?1?d?1?1?d?1?1?1?1: 01% (77937)
[+]  ?1?1?1?1?d?d?1?1?1?1: 01% (77717)
[+]  ?1?1?d?1?d?1?1?1?1?1: 01% (77618)
[+]  ?1?1?1?1?1?1?d?d?1?1: 01% (77403)
[+]  ?1?1?d?d?1?1?1?1?1?1: 01% (76936)
[+]  ?1?d?d?1?1?1?1?1?1?1: 01% (76649)
[+]  ?d?1?d?1?1?1?1?1?1?1: 01% (76570)
[+]  ?d?d?1?1?1?1?1?1?1?1: 01% (76489)
[+]  ?1?1?1?d?d?1?1?1?1?1: 01% (76477)
[+]  ?d?1?1?d?1?1?1?1?1?1: 01% (76166)
[+]  ?d?1?1?1?d?1?1?1?1?1: 01% (76159)
[+]  ?d?1?1?1?1?1?d?1?1?1: 01% (75954)
[+]          ?d?d?d?d?d?d: 01% (75450)
[+]  ?1?1?1?1?1?d?d?1?1?1: 01% (75429)
[+]  ?d?1?1?1?1?d?1?1?1?1: 01% (75381)
[+]  ?1?1?d?1?1?1?1?d?1?1: 01% (74433)
[+]      ?1?1?1?1?d?d?d?d: 01% (74278)
[+]  ?1?1?1?1?1?1?1?d?d?1: 01% (73645)
[+]        ?1?1?1?1?1?d?d: 01% (69360)
[+]  ?1?1?1?1?1?1?1?1?d?d: 01% (67987)
[+]    ?1?1?1?1?1?d?d?d?d: 01% (65773)

[*] Saving Mask statistics to ../../analysis/pack/masks/gamigo.
  csv
```

Appendix G

Pack - Linkedin

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
    faster parsing.
[*] Analyzing passwords: ../../lists/linkedin/passwords/sorted.
    txt
[+] Analyzing 100% (5403392/5403392) passwords
     NOTE: Statistics below is relative to the number of analyzed
          passwords, not total number of passwords

[*] Line Count Statistics...
[+]                             8: 31% (1727267)
[+]                             9: 17% (960775)
[+]                            10: 12% (690556)
[+]                             7: 11% (617401)
[+]                             6: 10% (584877)
[+]                            11: 06% (364901)
[+]                            12: 04% (222753)
[+]                            13: 02% (114421)
[+]                            14: 01% (67023)

[*] Mask statistics...
[+]                   stringdigit: 38% (2083937)
[+]                     allstring: 24% (1322955)
[+]                     othermask: 12% (678398)
[+]             stringdigitstring: 10% (566169)
[+]                   digitstring: 04% (222746)
[+]                      alldigit: 03% (202589)
[+]            stringspecialdigit: 02% (144440)
[+]              digitstringdigit: 01% (85121)
[+]           stringspecialstring: 01% (62467)
[+]                 stringspecial: 00% (26258)
[+]                 specialstring: 00% (4412)
[+]          specialstringspecial: 00% (3161)
[+]                    allspecial: 00% (739)
```

```
[*] Charset statistics...
[+]              loweralphanum: 44% (2387265)
[+]                 loweralpha: 21% (1153838)
[+]              mixedalphanum: 15% (848051)
[+]       loweralphaspecialnum: 04% (230962)
[+]       mixedalphaspecialnum: 04% (222184)
[+]                    numeric: 03% (202589)
[+]                 mixedalpha: 02% (138598)
[+]          loweralphaspecial: 01% (71265)
[+]              upperalphanum: 01% (68998)
[+]          mixedalphaspecial: 00% (33156)
[+]                 upperalpha: 00% (30519)
[+]       upperalphaspecialnum: 00% (12866)
[+]          upperalphaspecial: 00% (2362)
[+]                    special: 00% (739)

[*] Advanced Mask statistics...
[+]           ?l?l?l?l?l?l?l?l: 04% (247933)
[+]           ?l?l?l?l?l?l?d?d: 03% (191913)
[+]         ?l?l?l?l?l?l?l?l?l: 03% (179935)
[+]               ?l?l?l?l?l?l: 03% (175640)
[+]       ?l?l?l?l?l?l?l?l?l?l: 02% (158504)
[+]             ?l?l?l?l?l?l?l: 02% (149567)
[+]           ?l?l?l?l?d?d?d?d: 01% (107497)
[+]     ?l?l?l?l?l?l?l?l?l?l?l: 01% (99590)
[+]               ?d?d?d?d?d?d: 01% (93337)
[+]           ?l?l?l?l?l?l?l?d: 01% (87642)
[+]         ?l?l?l?l?l?l?l?d?d: 01% (85959)
[+]             ?l?l?l?l?l?d?d: 01% (76002)
[+]         ?l?l?l?l?l?d?d?d?d: 01% (71694)
[+]   ?l?l?l?l?l?l?l?l?l?l?l?l: 01% (64981)
[+]           ?l?l?l?l?l?d?d?d: 01% (63866)
[+]       ?l?l?l?l?l?l?d?d?d?d: 01% (61317)
[+]       ?l?l?l?l?l?l?l?l?d?d: 01% (59911)
[+]             ?l?l?l?d?d?d?d: 01% (55941)
[+]           ?d?d?d?d?d?d?d?d: 01% (55026)

[*] Saving Mask statistics to ../../analysis/pack/masks/linkedin
    .csv
```

Appendix H

Pack - phpBB

```
[?] Psyco is not available. Install Psyco on 32−bit systems for
   faster parsing.
[∗] Analyzing passwords: ../../lists/phpbb/passwords/sorted.txt
[+] Analyzing 100% (256288/256288) passwords
    NOTE: Statistics below is relative to the number of analyzed
        passwords, not total number of passwords

[∗] Line Count Statistics...
[+]                        6: 27% (69516)
[+]                        8: 26% (68111)
[+]                        7: 17% (45167)
[+]                        9: 09% (23498)
[+]                        5: 05% (14478)
[+]                       10: 05% (14197)
[+]                        4: 03% (8070)
[+]                       11: 02% (5705)
[+]                       12: 01% (3083)

[∗] Mask statistics...
[+]                allstring: 52% (135708)
[+]              stringdigit: 21% (53858)
[+]                 alldigit: 12% (30785)
[+]                othermask: 05% (13010)
[+]        stringdigitstring: 04% (12087)
[+]              digitstring: 02% (5999)
[+]         digitstringdigit: 00% (2280)
[+]      stringspecialstring: 00% (1084)
[+]       stringspecialdigit: 00% (608)
[+]            stringspecial: 00% (550)
[+]            specialstring: 00% (124)
[+]     specialstringspecial: 00% (111)
[+]               allspecial: 00% (84)

[∗] Charset statistics...
```

```
[+]              loweralpha: 50% (128459)
[+]           loweralphanum: 28% (73871)
[+]                 numeric: 12% (30785)
[+]           mixedalphanum: 03% (9071)
[+]              mixedalpha: 02% (5352)
[+]           upperalphanum: 00% (2060)
[+]              upperalpha: 00% (1897)
[+]    loweralphaspecialnum: 00% (1888)
[+]       loweralphaspecial: 00% (1670)
[+]    mixedalphaspecialnum: 00% (682)
[+]       mixedalphaspecial: 00% (358)
[+]                 special: 00% (84)
[+]    upperalphaspecialnum: 00% (64)
[+]       upperalphaspecial: 00% (47)

[*] Advanced Mask statistics...
[+]            ?l?l?l?l?l?l: 14% (37770)
[+]        ?l?l?l?l?l?l?l?l: 10% (27495)
[+]          ?l?l?l?l?l?l?l: 09% (24325)
[+]            ?d?d?d?d?d?d: 05% (13265)
[+]      ?l?l?l?l?l?l?l?l?l: 04% (10860)
[+]              ?l?l?l?l?l: 04% (10625)
[+]    ?l?l?l?l?l?l?l?l?l?l: 02% (6322)
[+]        ?d?d?d?d?d?d?d?d: 02% (6251)
[+]        ?l?l?l?l?l?l?d?d: 02% (5407)
[+]                ?l?l?l?l: 01% (4511)
[+]          ?l?l?l?l?l?d?d: 01% (3449)
[+]          ?d?d?d?d?d?d?d: 01% (3249)
[+]        ?l?l?l?l?d?d?d?d: 01% (3246)
[+]            ?l?l?l?l?l?d: 01% (3209)
[+]            ?l?l?l?l?d?d: 01% (3077)
[+]        ?l?l?l?l?l?l?l?d: 01% (3065)
[+]                ?d?d?d?d: 01% (2999)
[+]        ?l?l?l?l?l?d?d?d: 01% (2777)
[+]  ?l?l?l?l?l?l?l?l?l?l?l: 01% (2735)

[*] Saving Mask statistics to ../../analysis/pack/masks/phpbb.
   csv
```

Appendix I

Pack - RockYou

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
    faster parsing.
[*] Analyzing passwords: ../../lists/rockyou/passwords/sorted.
    txt
[+] Analyzing 100% (32602639/32602639) passwords
    NOTE: Statistics below is relative to the number of analyzed
          passwords, not total number of passwords

[*] Line Count Statistics...
[+]                              6: 26% (8488359)
[+]                              8: 19% (6513033)
[+]                              7: 19% (6287948)
[+]                              9: 12% (3949755)
[+]                             10: 09% (2954593)
[+]                              5: 04% (1326950)
[+]                             11: 03% (1163241)
[+]                             12: 02% (686824)
[+]                             13: 01% (429673)

[*] Mask statistics...
[+]                      allstring: 44% (14359391)
[+]                    stringdigit: 30% (9833044)
[+]                       alldigit: 15% (5193093)
[+]                    digitstring: 02% (895678)
[+]                      othermask: 02% (829934)
[+]              stringdigitstring: 01% (597382)
[+]            stringspecialstring: 00% (258197)
[+]                  stringspecial: 00% (241778)
[+]             stringspecialdigit: 00% (189675)
[+]               digitstringdigit: 00% (148099)
[+]           specialstringspecial: 00% (30202)
[+]                  specialstring: 00% (16641)
[+]                     allspecial: 00% (9525)
```

```
[*] Charset statistics...
[+]                lowerralpha: 41% (13589764)
[+]             loweralphanum: 33% (10814848)
[+]                   numeric: 15% (5193093)
[+]        loweralphaspecial: 01% (534600)
[+]             upperalphanum: 01% (532265)
[+]     loweralphaspecialnum: 01% (522358)
[+]                upperalpha: 01% (488177)
[+]             mixedalphanum: 01% (463520)
[+]                mixedalpha: 00% (281450)
[+]     mixedalphaspecialnum: 00% (57747)
[+]        mixedalphaspecial: 00% (55771)
[+]        upperalphaspecial: 00% (30406)
[+]     upperalphaspecialnum: 00% (29115)
[+]                   special: 00% (9525)

[*] Advanced Mask statistics...
[+]             ?l?l?l?l?l?l: 12% (3987972)
[+]           ?l?l?l?l?l?l?l: 08% (2738090)
[+]         ?l?l?l?l?l?l?l?l: 07% (2469728)
[+]             ?d?d?d?d?d?d: 06% (2278937)
[+]       ?l?l?l?l?l?l?l?l?l: 04% (1382192)
[+]               ?l?l?l?l?l: 02% (943285)
[+]         ?l?l?l?l?l?l?d?d: 02% (923992)
[+]     ?l?l?l?l?l?l?l?l?l?l: 02% (869683)
[+]         ?d?d?d?d?d?d?d?d: 02% (817588)
[+]           ?l?l?l?l?l?d?d: 02% (751557)
[+]           ?l?l?l?l?l?l?d: 02% (726231)
[+]           ?d?d?d?d?d?d?d: 01% (641777)
[+]             ?l?l?l?l?d?d: 01% (614420)
[+]         ?l?l?l?l?l?l?l?d: 01% (568674)
[+]             ?l?l?l?l?l?d: 01% (540869)
[+]     ?d?d?d?d?d?d?d?d?d?d: 01% (540786)
[+]       ?l?l?l?l?l?l?l?d?d: 01% (488287)
[+]   ?l?l?l?l?l?l?l?l?l?l?l: 01% (456864)
[+]       ?d?d?d?d?d?d?d?d?d: 01% (449025)
[+]       ?l?l?l?l?l?l?l?l?d: 01% (401075)
[+]     ?l?l?l?l?l?l?l?l?d?d: 01% (356681)

[*] Saving Mask statistics to ../../analysis/pack/masks/rockyou.
    csv
```

Appendix J

Pack - Rootkit.com

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
    faster parsing.
[*] Analyzing passwords: ../../lists/rootkit.com/passwords/
    sorted.txt
[+] Analyzing 100% (66819/66819) passwords
     NOTE: Statistics below is relative to the number of analyzed
          passwords, not total number of passwords

[*] Line Count Statistics...
[+]                          8: 26% (17561)
[+]                          6: 25% (16897)
[+]                          7: 17% (11567)
[+]                          9: 10% (7236)
[+]                         10: 06% (4454)
[+]                          5: 04% (2682)
[+]                         11: 03% (2146)
[+]                          4: 02% (1444)
[+]                         12: 01% (1298)

[*] Mask statistics...
[+]                  allstring: 44% (30041)
[+]                stringdigit: 21% (14538)
[+]                   alldigit: 14% (9582)
[+]                  othermask: 07% (5343)
[+]          stringdigitstring: 05% (3384)
[+]                digitstring: 02% (1708)
[+]           digitstringdigit: 00% (655)
[+]        stringspecialstring: 00% (558)
[+]         stringspecialdigit: 00% (403)
[+]              stringspecial: 00% (351)
[+]              specialstring: 00% (111)
[+]       specialstringspecial: 00% (93)
[+]                 allspecial: 00% (52)
```

```
[*] Charset statistics...
[+]                 loweralpha: 42% (28548)
[+]              loweralphanum: 31% (20804)
[+]                    numeric: 14% (9582)
[+]              mixedalphanum: 03% (2560)
[+]       loweralphaspecialnum: 02% (1462)
[+]                 mixedalpha: 01% (1166)
[+]          loweralphaspecial: 01% (1033)
[+]       mixedalphaspecialnum: 00% (606)
[+]              upperalphanum: 00% (387)
[+]                 upperalpha: 00% (327)
[+]          mixedalphaspecial: 00% (233)
[+]                    special: 00% (52)
[+]       upperalphaspecialnum: 00% (38)
[+]          upperalphaspecial: 00% (21)

[*] Advanced Mask statistics...
[+]               ?l?l?l?l?l?l: 11% (7975)
[+]           ?l?l?l?l?l?l?l?l: 09% (6305)
[+]             ?l?l?l?l?l?l?l: 08% (5363)
[+]               ?d?d?d?d?d?d: 06% (4315)
[+]         ?l?l?l?l?l?l?l?l?l: 03% (2639)
[+]           ?d?d?d?d?d?d?d?d: 03% (2036)
[+]                 ?l?l?l?l?l: 02% (1816)
[+]       ?l?l?l?l?l?l?l?l?l?l: 02% (1722)
[+]           ?l?l?l?l?l?l?d?d: 01% (1330)
[+]             ?d?d?d?d?d?d?d: 01% (1197)
[+]           ?l?l?l?l?d?d?d?d: 01% (949)
[+]     ?l?l?l?l?l?l?l?l?l?l?l: 01% (798)
[+]                   ?l?l?l?l: 01% (755)
[+]           ?l?l?l?l?l?d?d?d: 01% (740)
[+]             ?l?l?l?l?l?d?d: 01% (730)
[+]           ?l?l?l?l?l?l?l?d: 01% (672)

[*] Saving Mask statistics to ../../analysis/pack/masks/rootkit.
    com.csv
```

Appendix K

Pack - Sony

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
   faster parsing.
[*] Analyzing passwords: ../../lists/sony/passwords/sorted.txt
[+] Analyzing 100% (36308/36308) passwords
    NOTE: Statistics below is relative to the number of analyzed
         passwords, not total number of passwords

[*] Line Count Statistics...
[+]                          8: 27% (10008)
[+]                          6: 27% (9925)
[+]                          7: 19% (7050)
[+]                          9: 11% (4269)
[+]                         10: 06% (2395)
[+]                         11: 02% (914)
[+]                          5: 01% (666)
[+]                         12: 01% (422)

[*] Mask statistics...
[+]                  allstring: 47% (17290)
[+]                stringdigit: 39% (14187)
[+]                   alldigit: 04% (1639)
[+]                digitstring: 03% (1152)
[+]          stringdigitstring: 02% (1048)
[+]                  othermask: 01% (618)
[+]           digitstringdigit: 00% (232)
[+]         stringspecialdigit: 00% (62)
[+]        stringspecialstring: 00% (41)
[+]              stringspecial: 00% (35)
[+]              specialstring: 00% (2)
[+]       specialstringspecial: 00% (1)
[+]                 allspecial: 00% (1)

[*] Charset statistics...
[+]                 loweralpha: 44% (16207)
[+]               loweralphanum: 42% (15267)
[+]                     numeric: 04% (1639)
[+]               mixedalphanum: 03% (1357)
[+]                  mixedalpha: 01% (656)
[+]               upperalphanum: 01% (458)
[+]                  upperalpha: 01% (427)
[+]        loweralphaspecialnum: 00% (118)
[+]        mixedalphaspecialnum: 00% (84)
[+]           loweralphaspecial: 00% (72)
[+]           mixedalphaspecial: 00% (15)
[+]        upperalphaspecialnum: 00% (6)
[+]           upperalphaspecial: 00% (1)
[+]                     special: 00% (1)

[*] Advanced Mask statistics...
[+]                ?l?l?l?l?l?l: 16% (5869)
[+]            ?l?l?l?l?l?l?l?l: 09% (3554)
[+]              ?l?l?l?l?l?l?l: 09% (3516)
[+]            ?l?l?l?l?l?l?d?d: 05% (1863)
[+]          ?l?l?l?l?l?l?l?l?l: 03% (1328)
[+]            ?l?l?l?l?l?l?l?d: 02% (973)
[+]                ?l?l?l?l?l?d: 02% (931)
[+]                ?d?d?d?d?d?d: 02% (930)
[+]              ?l?l?l?l?l?d?d: 02% (873)
[+]              ?l?l?l?l?l?l?d: 02% (803)
[+]            ?l?l?l?l?d?d?d?d: 02% (737)
[+]        ?l?l?l?l?l?l?l?l?l?l: 01% (690)
[+]          ?l?l?l?l?l?l?l?d?d: 01% (663)
[+]            ?l?l?l?l?l?d?d?d: 01% (612)
[+]                ?l?l?l?l?d?d: 01% (606)
[+]                  ?l?l?l?l?l: 01% (545)
[+]          ?l?l?l?l?l?l?l?l?d: 01% (488)
[+]          ?l?l?l?l?l?d?d?d?d: 01% (474)
[+]        ?l?l?l?l?l?l?l?l?d?d: 01% (419)
[+]              ?l?l?l?d?d?d?d: 01% (382)

[*] Saving Mask statistics to ../../analysis/pack/masks/sony.csv
```

Appendix L

Pack - Yahoo

```
[?] Psyco is not available. Install Psyco on 32-bit systems for
    faster parsing.
[*] Analyzing passwords:  ../../lists/yahoo/passwords/sorted.txt
[+] Analyzing 100% (442838/442838) passwords
     NOTE: Statistics below is relative to the number of analyzed
           passwords, not total number of passwords

[*] Line Count Statistics...
[+]                           8: 26% (119134)
[+]                           6: 17% (79628)
[+]                           9: 14% (65968)
[+]                           7: 14% (65608)
[+]                          10: 12% (54761)
[+]                          12: 04% (21729)
[+]                          11: 04% (21219)
[+]                           5: 01% (5322)

[*] Mask statistics...
[+]                 stringdigit: 41% (185311)
[+]                   allstring: 34% (153410)
[+]                    alldigit: 05% (26080)
[+]                  othermask: 05% (25039)
[+]                 digitstring: 05% (24962)
[+]           stringdigitstring: 04% (18676)
[+]            digitstringdigit: 01% (4647)
[+]          stringspecialdigit: 00% (2362)
[+]               stringspecial: 00% (1112)
[+]         stringspecialstring: 00% (927)
[+]        specialstringspecial: 00% (168)
[+]               specialstring: 00% (127)
[+]                  allspecial: 00% (17)

[*] Charset statistics...
[+]              loweralphanum: 50% (224086)
[+]                 loweralpha: 33% (146512)
[+]                    numeric: 05% (26080)
[+]              mixedalphanum: 05% (23233)
[+]      loweralphaspecialnum: 01% (6289)
[+]                 mixedalpha: 01% (5122)
[+]              upperalphanum: 00% (3416)
[+]      mixedalphaspecialnum: 00% (3349)
[+]         loweralphaspecial: 00% (2190)
[+]                 upperalpha: 00% (1776)
[+]          mixedalphaspecial: 00% (496)
[+]      upperalphaspecialnum: 00% (223)
[+]         upperalphaspecial: 00% (49)
[+]                    special: 00% (17)

[*] Advanced Mask statistics...
[+]              ?l?l?l?l?l?l: 09% (40693)
[+]          ?l?l?l?l?l?l?l?l: 07% (32438)
[+]            ?l?l?l?l?l?l?l: 06% (29129)
[+]          ?l?l?l?l?l?l?d?d: 04% (20315)
[+]        ?l?l?l?l?l?l?l?l?l: 03% (16185)
[+]      ?l?l?l?l?l?l?l?l?d?d: 02% (12631)
[+]              ?d?d?d?d?d?d: 02% (12583)
[+]          ?l?l?l?l?l?l?l?d: 02% (10620)
[+]      ?l?l?l?l?l?l?l?l?l?l: 02% (10310)
[+]        ?l?l?l?l?l?l?l?d?d: 02% (10281)
[+]          ?l?l?l?l?d?d?d?d: 01% (8394)
[+]            ?l?l?l?l?l?d?d: 01% (8174)
[+]          ?l?l?l?l?l?d?d?d: 01% (7123)
[+]            ?l?l?l?l?l?l?d: 01% (7122)
[+]              ?l?l?l?l?l?d: 01% (6452)
[+]        ?l?l?l?l?l?l?d?d?d: 01% (6109)
[+]        ?l?l?l?l?l?l?l?l?d: 01% (6053)
[+]      ?l?l?l?l?l?l?d?d?d?d: 01% (5880)
[+]              ?l?l?l?l?d?d: 01% (5756)
[+]        ?l?l?l?l?l?d?d?d?d: 01% (5701)
[+]          ?d?d?d?d?d?d?d?d: 01% (5285)
[+]    ?l?l?l?l?l?l?l?l?l?l?l: 01% (4964)
[+]  ?l?l?l?l?l?l?l?l?l?l?l?l: 01% (4888)

[*] Saving Mask statistics to ../../analysis/pack/masks/yahoo.
    csv
```

Appendix M

Cracking System Benchmarks

Hash Algorithm	Speed (Passwords / Second)
MD5	13006.5M/s
Joomla	10658.0M/s
SHA1	5533.9M/s
MSSQL(2000)	4885.3M/s
phpass, MD5(Wordpress), MD5(phpBB3)	5482.2k/s
md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5	8954.3k/s
NThash, NTLM	17183.5M/s
Domain Cached Credentials, mscash	9302.1M/s
SHA256	2246.0M/s
descrypt, DES(Unix), Traditional DES	154.7M/s
SHA512	242.9M/s
sha512crypt, SHA512(Unix)	33174/s
LM	2382.1M/s
Oracle 7-10g, DES(Oracle)	570.6M/s
bcrypt, Blowfish(OpenBSD)	8519/s

Table M.1:

Appendix N

Masks only occurring in one dataset

gamigo	?l?l?l?u?u?u?d?d?d
rockyou	?d?d?d?d?d?d?d?d?d?d
rockyou	?d?d?d?d?d?d?d?d?d
gamigo	?l?l?d?l?l?d?l?l?l?l
gamigo	?l?l?l?l?d?d?l?l?l?l
gamigo	?l?l?d?l?d?l?l?l?l?l
gamigo	?l?l?l?l?l?l?d?d?l?l
gamigo	?l?l?d?d?l?l?l?l?l?l
gamigo	?l?d?d?l?l?l?l?l?l?l
gamigo	?d?l?d?l?l?l?l?l?l?l
gamigo	?d?d?l?l?l?l?l?l?l?l
gamigo	?l?l?l?d?d?l?l?l?l?l
gamigo	?d?l?l?d?l?l?l?l?l?l
gamigo	?d?l?l?l?d?l?l?l?l?l
gamigo	?d?l?l?l?l?l?d?l?l?l
battlefieldheroes	?l?l?l?l?d?d?d
gamigo	?l?l?l?l?l?d?d?l?l?l
gamigo	?d?l?l?l?l?d?l?l?l?l
gamigo	?l?l?d?l?l?l?l?d?l?l
phpbb	?d?d?d?d
gamigo	?l?l?l?l?l?l?l?d?d?l

Table N.1:

Appendix O

Masks occurring in more than one dataset

Mask	Datasets	Percentage	Cumulative percentage	σ	Key space	Cracking time per mask (days)
?l?l?l?l	2	1.45	1.45	0.32	456,976	0.00
?d?d?d?d?d?d	10	4.4	5.85	2.74	1,000,000	0.00
?d?d?d?d?d?d?d	5	1.74	7.59	0.26	10,000,000	0.00
?l?l?l?l?l	5	2.64	10.23	0.9	11,881,376	0.00
?l?l?l?d?d?d	2	1.27	11.5	0.2	17,576,000	0.00
?l?l?l?l?d?d	6	1.62	13.12	0.31	45,697,600	0.00
?d?d?d?d?d?d?d?d	9	2.54	15.66	1.5	100,000,000	0.00
?l?l?l?l?l?d	6	1.79	17.45	0.42	118,813,760	0.00
?l?l?l?d?d?d?d	3	1.13	18.58	0.12	175,760,000	0.00
?l?l?l?l?l?l	10	9.87	28.45	4.6	308,915,776	0.00
?l?l?l?l?l?d?d	10	1.72	30.17	0.59	1,188,137,600	0.00
?l?l?l?l?l?l?d	6	1.71	31.88	0.48	3,089,157,760	0.00
?l?l?l?l?d?d?d?d	9	1.77	33.65	0.5	4,569,760,000	0.00
?l?l?l?l?l?l?l	10	6.76	40.41	2.7	8,031,810,176	0.00
?l?l?l?l?l?d?d?d	7	1.68	42.09	0.63	11,881,376,000	0.00
?l?l?l?l?l?l?d?d	10	3.7	45.79	1.6	30,891,577,600	0.00
?l?l?l?l?l?l?l?d	9	1.88	47.67	0.69	80,318,101,760	0.00
?l?l?l?l?l?d?d?d?d	5	1.2	48.87	0.13	118,813,760,000	0.00
?l?l?l?l?l?l?l?l	10	7.71	56.58	2.35	208,827,064,576	0.00
?l?l?l?l?l?l?d?d?d	2	1.59	58.17	0.21	308,915,776,000	0.00
?l?l?l?l?l?l?l?d?d	7	1.6	59.77	0.36	803,181,017,600	0.00
?l?l?l?l?l?l?l?l?d	4	1.25	61.02	0.12	2,088,270,645,760	0.00
?l?l?l?l?l?l?d?d?d?d	2	1.23	62.25	0.1	3,089,157,760,000	0.00
?l?l?l?l?l?l?l?l?l	10	3.48	65.73	0.77	5,429,503,678,976	0.00
?l?l?l?l?l?l?l?l?d?d	5	1.46	67.19	0.7	20,882,706,457,600	0.02
?l?l?l?l?l?l?l?l?l?l	9	2.28	69.47	0.46	141,167,095,653,376	0.13
?l?l?l?l?l?l?l?l?l?l?l	6	1.31	70.78	0.26	3,670,344,486,987,780	3.27
?l?l?l?l?l?l?l?l?l?l?l?l	4	1.32	72.1	0.18	95,428,956,661,682,200	84.96

Table O.1:

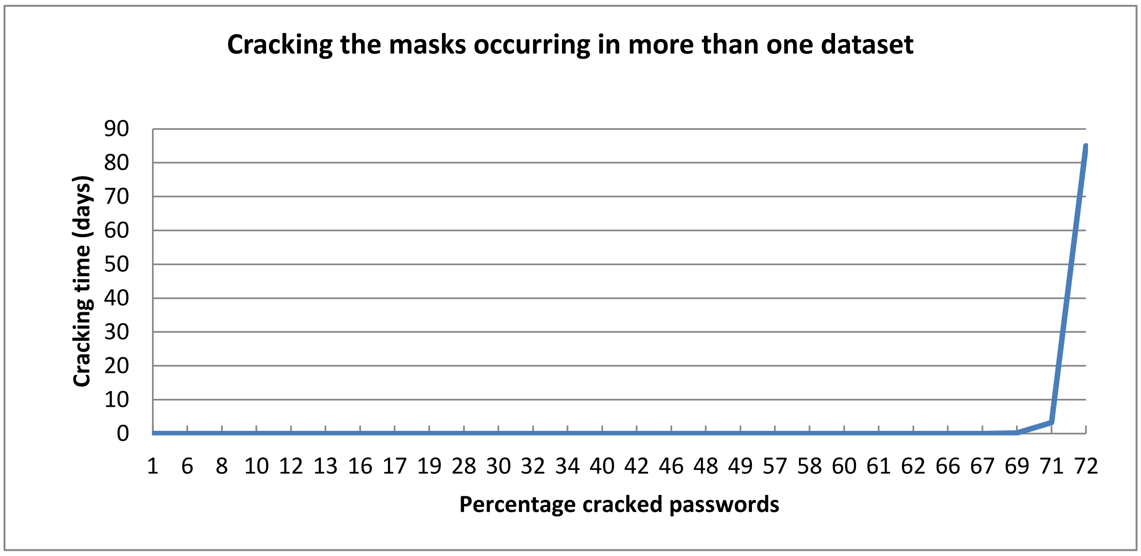


Figure O.1: Cracking the masks occurring in more than one dataset

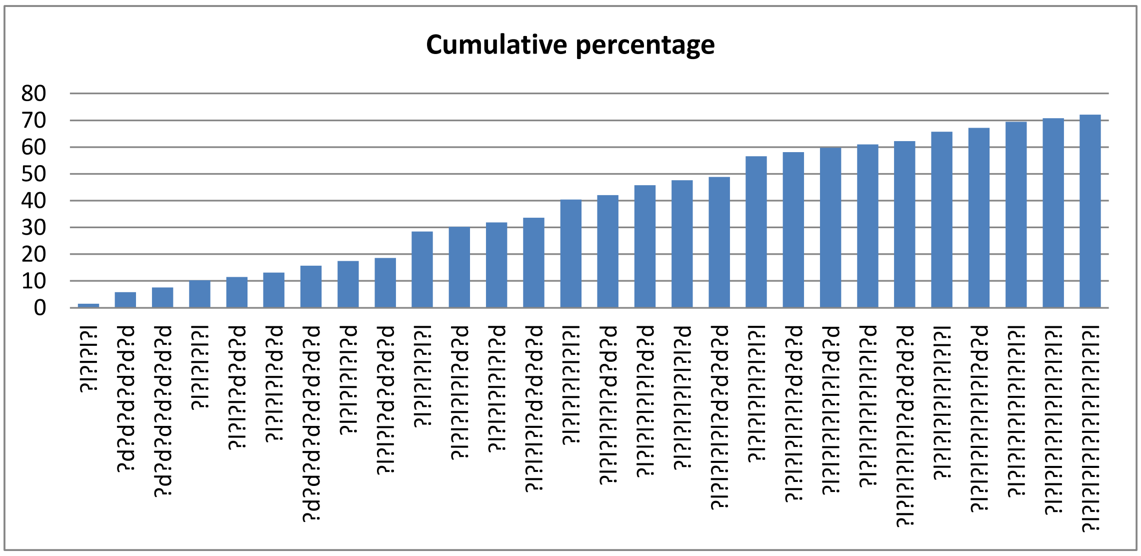


Figure O.2: Cracking the masks occurring in more than one dataset cumulative percentage

Benchmark passwords/second	Total percentage	Total key space	Days to crack
13,000,000,000	72.1	99,273,538,380,506,200	88.38

Table O.2:

Appendix P

Masks occurring in more than one dataset - without three largest masks

Mask	Datasets	Percentage	Cumulative percentage	σ	Key space	Cracking time per mask (minutes)
?l?l?l?l	2	1.45	1.45	0.32	456,976	0.00
?d?d?d?d?d?d	10	4.4	5.85	2.74	1,000,000	0.00
?d?d?d?d?d?d?d	5	1.74	7.59	0.26	10,000,000	0.00
?l?l?l?l?l	5	2.64	10.23	0.9	11,881,376	0.00
?l?l?l?d?d?d	2	1.27	11.5	0.2	17,576,000	0.00
?l?l?l?l?d?d	6	1.62	13.12	0.31	45,697,600	0.00
?d?d?d?d?d?d?d?d	9	2.54	15.66	1.5	100,000,000	0.00
?l?l?l?l?l?d	6	1.79	17.45	0.42	118,813,760	0.00
?l?l?l?d?d?d?d	3	1.13	18.58	0.12	175,760,000	0.00
?l?l?l?l?l?l	10	9.87	28.45	4.6	308,915,776	0.00
?l?l?l?l?l?d?d	10	1.72	30.17	0.59	1,188,137,600	0.00
?l?l?l?l?l?l?d	6	1.71	31.88	0.48	3,089,157,760	0.00
?l?l?l?l?d?d?d?d	9	1.77	33.65	0.5	4,569,760,000	0.01
?l?l?l?l?l?l?l	10	6.76	40.41	2.7	8,031,810,176	0.01
?l?l?l?l?l?d?d?d	7	1.68	42.09	0.63	11,881,376,000	0.02
?l?l?l?l?l?l?d?d	10	3.7	45.79	1.6	30,891,577,600	0.04
?l?l?l?l?l?l?l?d	9	1.88	47.67	0.69	80,318,101,760	0.10
?l?l?l?l?l?d?d?d?d	5	1.2	48.87	0.13	118,813,760,000	0.15
?l?l?l?l?l?l?l?l	10	7.71	56.58	2.35	208,827,064,576	0.27
?l?l?l?l?l?l?d?d?d	2	1.59	58.17	0.21	308,915,776,000	0.40
?l?l?l?l?l?l?l?d?d	7	1.6	59.77	0.36	803,181,017,600	1.03
?l?l?l?l?l?l?l?l?d	4	1.25	61.02	0.12	2,088,270,645,760	2.68
?l?l?l?l?l?l?d?d?d?d	2	1.23	62.25	0.1	3,089,157,760,000	3.96
?l?l?l?l?l?l?l?l?l	10	3.48	65.73	0.77	5,429,503,678,976	6.96
?l?l?l?l?l?l?l?l?d?d	5	1.46	67.19	0.7	20,882,706,457,600	26.77

Table P.1:

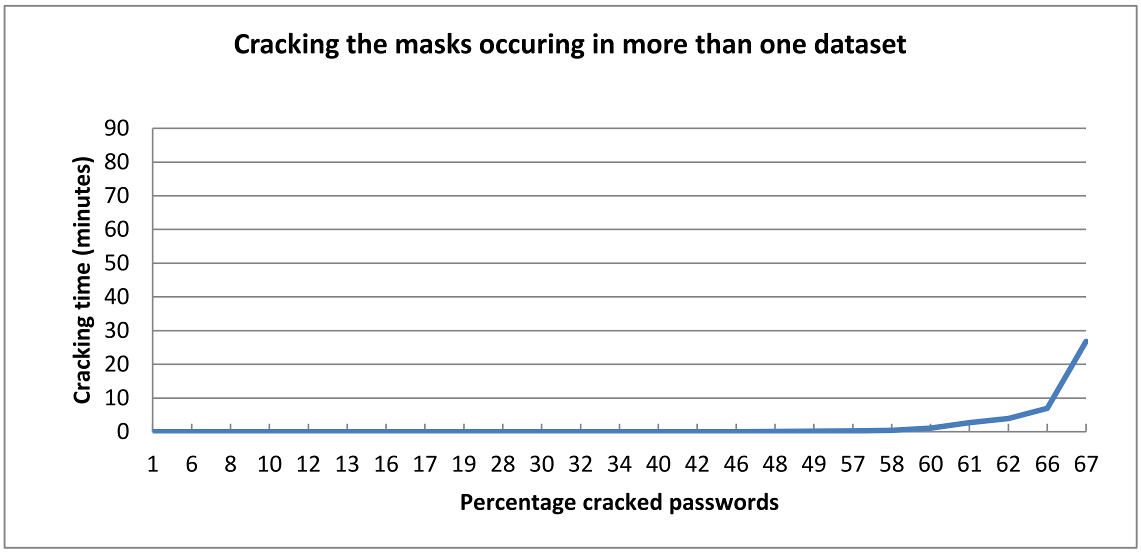


Figure P.1: Cracking the masks occurring in more than one dataset without the three largest masks

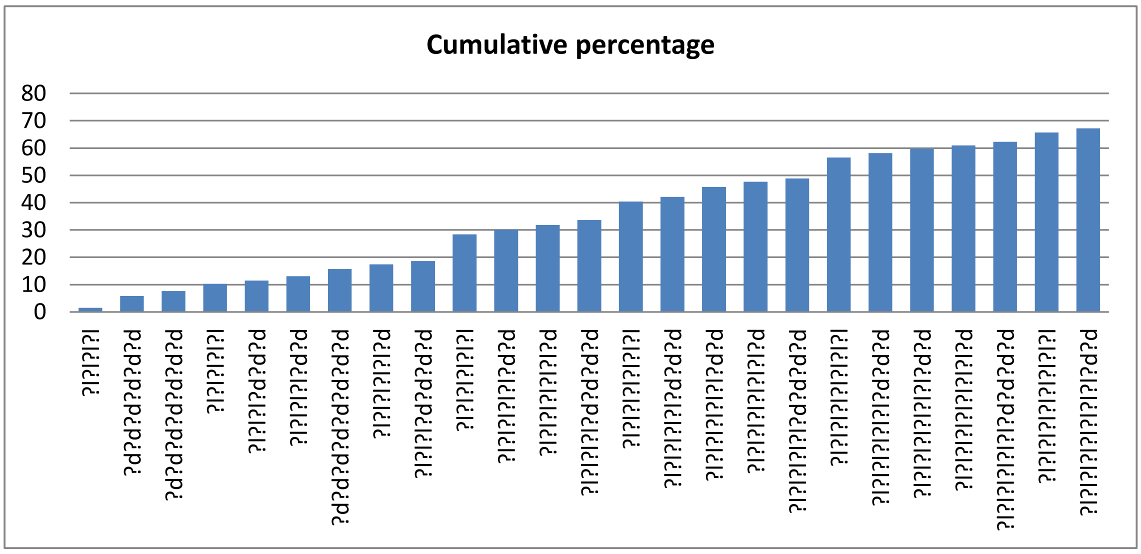


Figure P.2: Cracking the masks occurring in more than one dataset cumulative percentage without the three largest masks

Benchmark passwords/second	Total percentage	Total key space	Minutes to crack
13,000,000,000	67.19	33,070,136,182,896	42.40

Table P.2:

Appendix Q

Comparison of systems

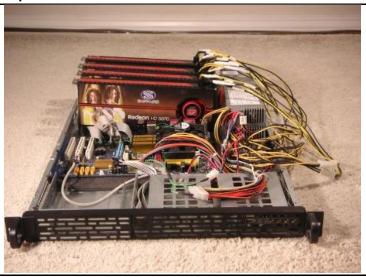
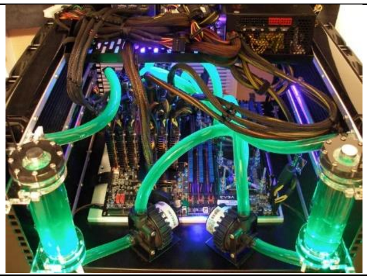
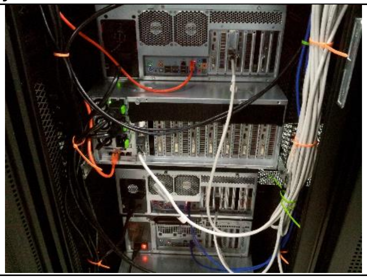
	Whitepixel	D3ad0ne - Project Erebus v2.5	Epixoip	TNO
Software	Whitepixel V2	oclHashcat-lite v0.10	oclHashcat-lite v0.11	oclHashcat-lite-0.15
Hardware	4x HD 5970 (dual GPU)	8x HD 7970	10x HD 7970 4x HD 5970 (dual GPU) 3x HD 6990 (dual GPU) 1x HD 5870	2x HD 6970 1x HD 7970 1x I7 3820 1x OCZ Vector 256GB 2x 1TB HDD
Speed (MD5 password/second)	33.1 billion	74.2 billion	180 billion	16 billion
Cost	< $3,000	> $10,000	> $20,000	< $2,000
Date	December 14, 2010	July 17, 2012	December 3, 2012	April 18, 2013
Power consumption	-	-	7 kW	< 800 Watt
Remark	*The oldest system. The software patched optimizations directly in the OpenCL machine code.*	*Uses SSDs with rainbow tables and dictionaries.*	*The only distributed GPGPU password hash cracking solution found.*	*Our system.*
				

Figure Q.1:

Appendix R

MD5 pseudocode

```
//MD5 Mainloop:
for i from 0 to 63
  if 0 ≤ i ≤ 15 then
    F := (B and C) or ((not B) and D) ← bitselect
    g := i
  else if 16 ≤ i ≤ 31
    F := (D and B)or ((not D) and C) ← bitselect
    g := (5∗i+1)mod16
  else if 32 ≤ i ≤ 47
    F := B xor C xor D
    g := (3∗i+5) mod 16
  else if 48 ≤ i ≤ 63
    F := C xor (B or (notD))
    g := (7∗i) mod 16
  dTemp := D
  D := C
  C := B
  B := B + leftrotate((A+F+K[i]+M[g]),s[i]) ← bitshift
  A := dTemp
endfor
```

Appendix S

Hashcat’s default bruteforce mask increment mode

User specific charset:	?1	?2	?3	?d
Hashcat charset:	?l?d?u	?l?d	?l?d*!$@_	?d
Key space	62	36	41	10
Benchmark used (passwords/second)	13,000,000,000			
Time to bruteforce complete mask (years)	2871			

Table S.1:

Masks	Keyspace	Crackingtimepermask(minutes)	Cumulativecrackingtime
?1	62	0,00	0,00
?1?2	2232	0,00	0,00
?1?2?2	80352	0,00	0,00
?1?2?2?2	2892672	0,00	0,00
?1?2?2?2?2	104136192	0,00	0,00
?1?2?2?2?2?2	3748902912	0,00	0,00
?1?2?2?2?2?2?2	153705019392	0,20	0,20
?1?2?2?2?2?2?2?3	6301905795072	8,08	8,28
?1?2?2?2?2?2?2?3?3	258378137597952	331,25	339,54
?1?2?2?2?2?2?2?3?3?3	10593503641516000	13581,41	13920,95
?1?2?2?2?2?2?2?3?3?3?3	105935036415160000	135814,15	149735,10
?1?2?2?2?2?2?2?3?3?3?3?d?	1059350364151600000	1358141,49	1507876,59
?1?2?2?2?2?2?2?3?3?3?3?d?d	10593503641516000000	13581414,93	15089291,52
?1?2?2?2?2?2?2?3?3?3?3?d?d?d	105935036415160000000	135814149,25	150903440,77
?1?2?2?2?2?2?2?3?3?3?3?d?d?d?d	1059350364151600000000	1358141492,50	1509044933,27

Table S.2:

Appendix T

Per position letter frequency heatmaps

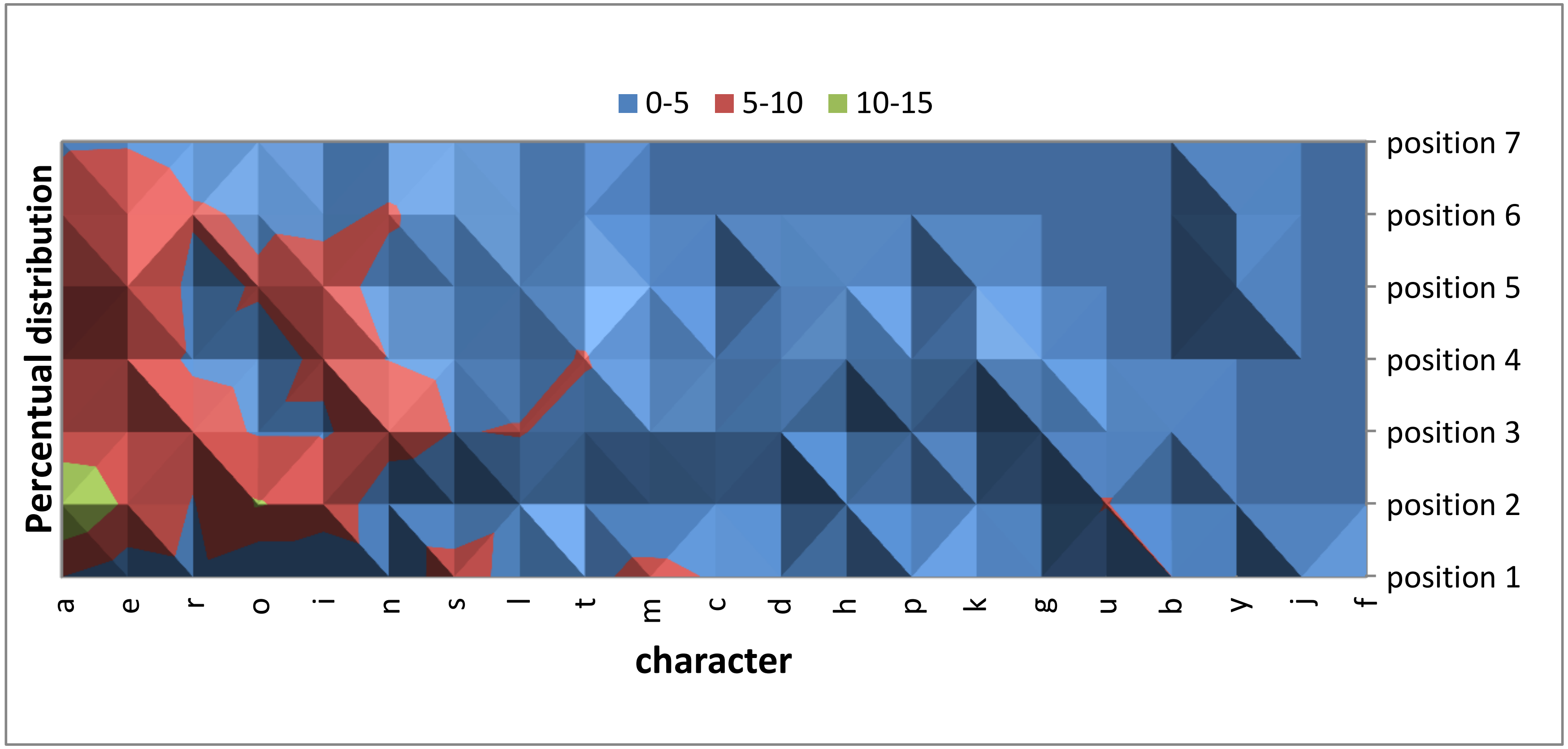


Figure T.1: The per position character distribution of passwords

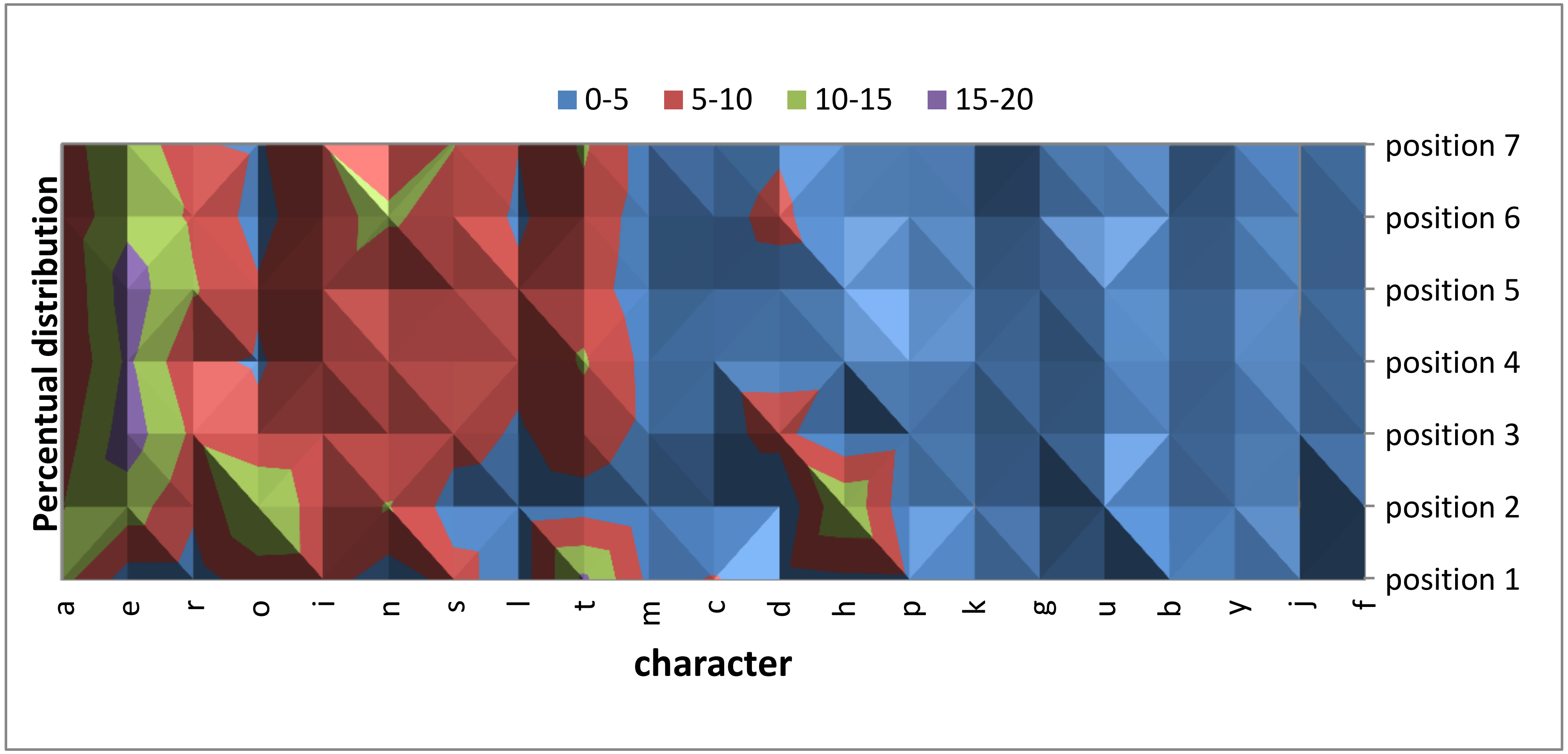


Figure T.2: The per position character distribution of English text