UNIVERSITY OF THE AEGEAN
SCHOOL OF ENGINEERING
DEPARTMENT OF INFORMATION AND COMMUNICATION SYSTEMS ENGINEERING

A study on strength estimation of non-English and non-ASCII passwords.

MASTER THESIS
by
EVGENIA PATSONI

INFORMATION AND COMMUNICATION SYSTEMS SECURITY

Supervisor: Dr. Panagiotis Rizomiliotis

Thesis Committee Members:

Dr. Maria Karyda
Dr. Spyros Kokolakis
Dr. Panagiotis Rizomiliotis

Samos, September 2020

Table of Contents

Catalog of Figures

Catalog of Tables

Abstract

For decades, text-based passwords have been the primary means of authentication to computer systems, as well as account security and disk encryption. Thus, in order to prevent users from selecting passwords that are too easy for an adversary to crack, password composition policies and password strength meters have been widely studied and adopted. Research has already shown that certain password composition policies and password strength meters fulfill this purpose and perform better than others. However, we believe that these results are biased when it comes to scoring non-English passwords, even for password strength meters who seemingly contribute to the creation of stronger passwords, because the majority of them is based on leaked password sets of English speaking users as their training base.

In this dissertation, we study modern password-strength meters as well as their accuracy and effect. Furthermore, we turn our attention on the ways an attacker can carry out a targeted password guessing attack, taking advantage of the nationality of the users. We train two modern password strength meters with a Greeklish training set, and try to prove our initial belief that they tend to falsely score non-English passwords as strong, focusing specifically on Greeklish passwords. We also analyze non-ASCII character passwords to find out if they actually provide any additional levels of security. Finally, we show how modern password cracking tools perform against such passwords, while trying to crack passwords that contain letters from the Greek alphabet.

A few studies, based on determining the actual strength of non-English passwords as well as the password creation habits of non-English users, especially Chinese, have been carried out in the past. To our knowledge, however, this is the first study that focuses entirely on the strength of Greeklish passwords, i.e. passwords that contain Greek words transcribed in English, and not simply passwords created by Greek users.

1. Chapter 1: About password-based authentication

Password-based forms of authentication have been and still are in widespread use, for authenticating humans to online as well as offline computer systems. They are ubiquitous because passwords are easy to understand, easy to implement and are widely supported by the IT infrastructure. The death of passwords might have been announced many times, but the lack of perfect alternatives is a proof that passwords will probably be the predominant means of user authentication in the foreseeable future.

It is an undeniable fact however, that passwords are often responsible for account compromise if not strong enough, hence there is an ongoing debate among researchers about the usability and the security passwords actually provide. More specifically, researchers have reported for decades that users struggle to comply with password creation and management and thus, in most cases, fail to create secure passwords. In addition, they tend to partially or exactly reuse these passwords across multiple websites and accounts, which makes password cracking and guessing even easier for the adversaries.

Various password composition policies and password strength meters, are some of the most widely deployed precautions that try to protect users against such attacks.

1.1. Password composition policies

Password composition policies are the rules to which passwords must conform. In the user authentication field, there is a wide range of such policies which may require passwords to exceed a minimum length, contain uppercase letters and symbols, not contain dictionary words, etc. [3]. Some users try to get away with the simplest password that fulfills the requirements, while others follow the rules carefully and even go beyond the requirements.

While it is necessary to advise users in order to ensure that they pick strong passwords, usability studies have many times concluded that existing password composition policies do not always achieve their goal of stronger passwords, as they can sometimes result in weaker password distributions that are more susceptible to cracking. In addition, such policies also affect users' behavior. For example, certain password composition policies that result to more-difficult-to-predict passwords, may lead users to write down

their passwords or to become more averse to changing their passwords often because of the additional effort needed to memorize the new ones [3].

1.2. Password strength meters

Password strength meters (PSMs) are usually embedded in a registration or password update page of a website. Their main goal, is to evaluate changes made to the password field at the time of password creation and output its strength. They may or may not be intertwined with a password composition policy.

Password strength meters are extensively studied in Chapter 2.

2. Chapter 2: Password strength meters

As mentioned earlier, a password strength meter calculates and displays an estimation of a user-chosen password, and either helps or forces the user to create passwords that are strong enough to provide an acceptable level of security [1]. PSMs are based on the hypothesis that users who choose weak passwords, do so because they are actually unaware that their passwords are weak. Hence, they aim to make users more aware of weak passwords through the meter's feedback, with the expectation that they will choose a stronger one.

As Ur et al. [16] mentioned in their study of password creation in the presence of password meters, password strength indicators vary among websites and usually include bar-like meters that display strength, checkmark-or-x systems and text indicating invalid characters and too-short passwords. Sites with bar-like meters use either a progress-bar metaphor or a segmented-box. Examples of these indicators are shown in Figure 1.

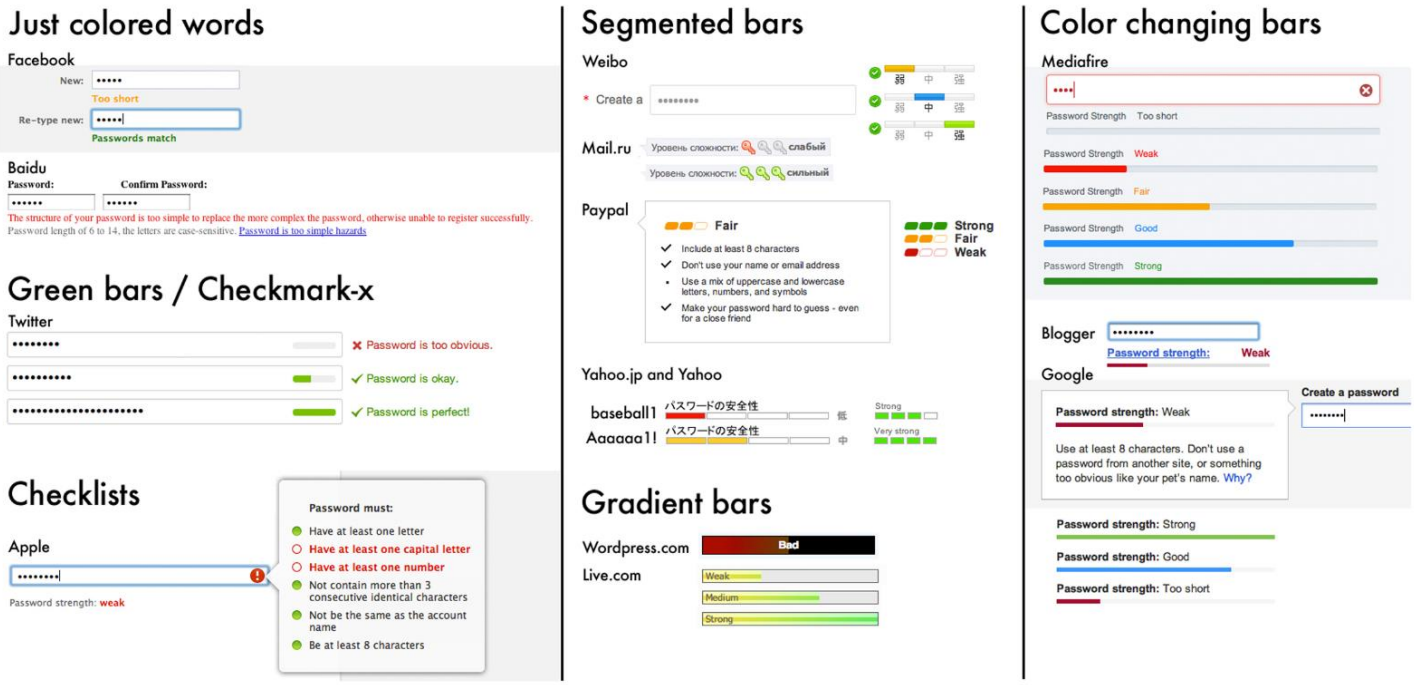

Figure 1: Password strength meters indicators examples [16]

Alike password composition policies, password strength meters do not always achieve their goal for stronger passwords. That is because a strength meter with low accuracy, may guide the user to create passwords with a high

score but low actual security. The accuracy of the most widely deployed PSMs, is studied later in this chapter.

2.1. Client and server-side password strength meters

Before moving on to the analysis of some of the most popular password strength meters, it is important to mention how they are usually deployed. Just like password encryption or validation etc., password strength estimation can either be deployed on the client-side or on the server-side.

Client-side password strength meters, calculate the strength of a password directly on the client's browser, using some JavaScript functions. Server-side password strength meters, send the password to a trusted, server-side checker through an AJAX request, without evaluating them locally.

2.2. Popular password strength metrics

Models of measuring password strength, often take one of two forms in theory. The first one relies on purely statistical methods, such as Shannon entropy or other advanced statistical approaches. The second one, relies on simulating adversarial password guessing techniques. Academic studies of password guessing, have mainly focused on probabilistic methods that take as input large, usually leaked password sets, and output guesses in descending probability order, while password cracking tools rely on efficient heuristics to model common password characteristics.

Golla et al. [1], examined some of the most popular password strength meters which have been proposed in academia, as well as meters which have real-life implementation (e.g. password managers, websites, etc.), and evaluated them as to their accuracy in measuring password strength. Some of the meters examined by the authors, along with one meter not present in their study, are analyzed below. Each one corresponds with a popular password strength metric.

"Heuristics / NIST Entropy"

In 2004 the NIST published SP 800-63 Ver.1.0 guideline [10], which included heuristics based on length and compliance to a composition policy on entropy estimation. The heuristic also considered a bonus if the password passed a common dictionary check.

Claude Shannon applied the term entropy to express, among other things, the amount of actual information in English text. He says, *"The entropy is a statistical parameter which measures in a certain sense, how much information is produced on the average for each letter of a text in the language. If the language is translated into binary digits (0 or 1) in the most efficient way, the entropy H is the average number of binary digits required per letter of the original language"*.

NIST gives a definition of entropy more oriented on password guessing that says, *"Entropy is a measure of the amount of uncertainty that an attacker faces to determine the value of a secret".* Entropy is usually measured in bits. If a password of k bits is chosen at random, there are 2^k possible values and the password is said to have k bits of entropy [10]. This means that the higher the guessing entropy of a password, the more difficult it is for the password to be guessed.

NIST entropy measurement is based on the fact that if we had a good knowledge of the frequency distribution of passwords chosen under a particular set of rules, then it would be straightforward to determine either the guessing entropy or the min-entropy of any password. Guessing entropy, is an estimation of the average amount of effort required to guess the password of a selected user, and the min-entropy is a measure of the difficulty of guessing the easiest single password in the population [10].

More specifically, NIST used the following rules to estimate guessing entropy. The entropy of the first character is taken to be 4 bits; The entropy of the next 7 characters are 2 bits per character; For the 9th through the 20th character the entropy is taken to be 1.5 bits per character; For characters 21 and above the entropy is taken to be 1 bit per character; A "bonus" of 6 bits of entropy is assigned for a composition rule that requires both upper case and non-alphabetic characters; A bonus of up to 6 bits of entropy is added for an extensive dictionary check.

For min-entropy estimation, they specified a dictionary test that was intended to ensure at least 10-bits of min entropy. That test is: Uppercase letters in passwords are converted to entirely lowercase and compared to a dictionary of at least 50.000 commonly selected, otherwise legal passwords and rejected if they match any dictionary entry; Passwords that are detectable permutations of the username are not allowed [10].

SP 800-63 Ver.1.0 password entropy estimation has been withdrawn for not being accurate enough and is superseded by more recent NIST publications.

Entropy though, is still used as a password strength metric in various researches.

"LUDS / Comprehensive8 & Basic16"

Traditionally, most password strength meters used to be based on ad-hoc approaches that merely count lowercase letters, uppercase letters, numbers of digits and symbols, mostly known as LUDS [1]. They do not take into account dictionary entries, common names, as well as known common passwords and their variants. Although it is now a well-known fact that LUDS approaches do not manage to measure the strength of a password accurately, experiments that have been designed to simulate password cracking scenarios, came to the conclusion that as the number of guesses increases - reaching one trillion - some LUDS based password composition policies are significantly hard to crack.

Comprehensive8, is a LUDS composition policy. More specifically, passwords that fall under this policy, should contain at least eight characters, including an uppercase letter, a lowercase letter, a symbol and a digit and should also not contain a dictionary word [2]. *Basic16* is another LUDS based composition policy, and passwords that fall under this policy should contain at least sixteen characters, with no further restrictions. In the same experiments as those mentioned above, at one trillion guesses *Basic16* passwords were cracked significantly less than any other condition. Moreover, after exhausting the Weir-algorithm guessing space, *Basic16* remained significantly hardest to crack [2]. NIST estimates that such sixteen-character passwords, have 48 bits of entropy against *Comprehensive8* passwords that have 30 bits of entropy [3]. Password strength meters based on LUDS, are usually deployed on the client-side.

"Password-Cracking Algorithms Simulation"

In 2012, Kelley et al. [2] proposed their implementation of measuring password strength by simulating password-cracking algorithms, while trying to better understand the effects of password-composition policies on the guessability of passwords. For this reason, the authors conducted an online study in order to compile a dataset of 12.000 plaintext passwords which were

collected from various participants under seven different password composition policies. Then, they developed a distributed technique: guess-number calculator, for calculating if and how long it would take for various password-guessing tools to guess each of the collected passwords, in other words a metric of password strength based on guessability.

A separate guess number calculator needs to be implemented for each cracking algorithm under consideration. In this work, the authors implemented two guess-number calculators: one for a brute-force algorithm loosely based on the Markov model (BFM calculator) and one for the heuristic algorithm proposed by Weir et al. Both algorithms require a training set, i.e. a corpus of known passwords used to generate a list of guesses and determine in what order they should be tried [2].

- *BFM calculator:* This algorithm is starting with the minimum length of the password policy and increases the length of guesses until all passwords are guessed, meaning that it covers the entire password space. More specifically, it processes the training data to generate a lookup table that maps each string to the number of guesses needed to reach it, as follows.

  For an alphabet of N characters and passwords of length L, if the first character tried does not match the first character of the target password, it is understood that the algorithm will try N^{L-1} incorrect guesses before switching to a different first character. So, if the first character of the password to be guessed is the k^{th} character to be tried, there will be at least $(k-1)^N{}^{L-1}$ incorrect guesses, and so forth. After looking up the order in which characters are tried, the number of incorrect guesses to discover how many iterations will be needed before a successful guess, without having to actually try the guesses, is summed up [2].

- *Weir algorithm calculator:* The Weir algorithm determines a finer-grained guessing order based on the probabilities of different password structures, or patterns of character types such as letters, digits, and symbols. It defines a terminal as one instantiation of a structure with specific substrings, and a probability group as a set of terminals with the same probability of occurrence.

  To calculate the guess number for a password, the calculator determines that password's probability group. Using the lookup table created from the training set, it determines the number of guesses

required to reach that probability group and then adds the number of guesses required to reach the exact password within that probability group [2].

"Markov Models / APSM"

Using Markov models to guess passwords, was first proposed in 2005 by Narayanan and Shmatikov [30]. Later in 2012, Castelluccia, et al. [4] proposed to use password strength meters based on Markov-models, which they claimed to estimate the true strength of a password more accurately than rule-based strength meters that measure the "entropy" of a password using various bonus and decrement rules. They call these Markov-models, Adaptive Password Strength Meters (APSMs), because they can react dynamically to changes on how users choose their passwords, meaning they take into account site-specific aspects of the password distribution. For their experiments, they made the assumptions that the adversary does not attack specific users, thus he does not use their personal information, but is rather interested in guessing any password in the system, and also that the adversary knows the distribution of the passwords.

The idea behind this, is that adjacent letters in passwords humans generate are not randomly or independently chosen, but usually follow certain regularities. This fact corresponds with the n-gram Markov model, where the probability of the next character in a string can be modelled based on a prefix of length n-1. For example, the 2-gram *th* is much more likely to occur in a password than the 2-gram *tq* and also the letter *e* usually comes after *th.* Thus, roughly speaking, the Markov-model estimates the strength of a password, by estimating the probability of the n-grams (i.e., frequencies of *n* consecutive characters in the password) that compose it. They also suggest that if the Markov-model is trained on the actual password database, even better results can be obtained.

In more detail, the authors stored the hashed and possibly salted value *Hash(password)*, for each new password that was added to the password list. In addition, they determined the n-gram counts for the password, merged this information with previously stored n-grams to obtain the frequencies over the entire password database, and then added some noise to this data in order to secure their construction against possible leakage of the n-gram database. These n-gram frequencies can later be used to estimate the probability of each

freshly registered password. Password Strength Meters based on *Markov-model* are usually deployed on the server-side [4].

Markov models have been widely analyzed and used in various works throughout the literature for example by Dürmuth et al. (2015), Golla et al. (2016), Angelstorf et al. (2017), etc.

"PCFG / fuzzyPSM"

In 2016, Wang et al. [6] proposed a fuzzy, probabilistic context-free password strength meter, which is based on a probabilistic context-free grammar (PCFG). A context-free grammar is a defined as $G = (V, \Sigma, S, P)$, where: V is a finite set of variables (or non-terminals), Σ is a finite set of terminals, S is the start variable, and P is a finite set of productions of the form: $\alpha \rightarrow \beta$, where α is a single variable and β is a string consisting of variables or terminals. Probabilistic context-free grammars simply have probabilities associated with each production, such that for a specific variable all the associated productions add up to 1 [29].

The idea behind PCFGs in terms of password creation, is that passwords are built based on template structures (e.g. 6 letters followed by 2 digits) and terminals that fit into those structures, so the probability of a password is the probability of its structure multiplied by the probabilities of its terminals. When working with PCFGs, the issue of grammar ambiguity should always be carefully considered. Grammar ambiguity arises when several base structures can produce the same terminal string. Generating guesses in highest probability order using a context-free grammar, relies on the grammar being unambiguous, which ensures that there is a well-defined probability for a guess that depends on a single, unique derivation. Otherwise, PCFG will still generate guesses for all base structures but probability order of guesses will not be preserved.

Wang et al. [6] research was mostly oriented on real user behavior. For this reason, the authors conducted a survey to prove the fact that users do not create a password from scratch each time they register for a service, but in most cases they fully or partially reuse older passwords. This observation has great implications for measuring password strength, because the majority of a user's passwords for a new account would be similar or same as their existing passwords and there is a high probability that an attacker already has knowledge of the latter.

To model password reuse behaviors, the authors employed passwords leaked from a less sensitive service as the base dictionary $\mathscr{B}$, to construct a basic password parsing-trie tree and another list of relatively strong passwords which were leaked from a sensitive service, as the training dictionary $\mathcal{T}$. Then, they parsed each password in the training dictionary and learned which and how mangling rules are employed by users to construct passwords for new services. This process, automatically creates a fuzzy probabilistic context-free grammar (PCFG) which is the base of the fuzzy-PCFG-based meter, fuzzyPSM.

In more detail, this meter includes three phases: training, measuring and update. The training phase can automatically derive all the observed base structures, base segments (including basic passwords and other segments) and transformation operations of all passwords in the training set $\mathcal{T}$ and their associated probabilities of occurrence. This builds a probabilistic context-free grammar that can later be used to measure password strength; that is, estimate the probabilities of occurrence of a given password in the measuring phase. The update phase, is used to capture the dynamic changes in the distribution of passwords in a system as time goes on, in order to bring the PCFG model closer to the real password distribution, resulting in an adaptive meter [6].

"Neural Networks"

In 2016, Melicher et al. [9] proposed the use of artificial neural networks for probabilistic password modeling. Artificial neural networks, are a machine-learning technique designed to model human neurons and to approximate highly dimensional functions. According to the authors, they are a natural fit for generating password guesses. The authors also implemented and benchmarked a client-side, neural-network password checker that gives real-time feedback on password strength and measures resistance to guessing attacks.

Similar to Markov models, the considered neural networks were trained to generate the next character of a password, given the preceding characters of it. This approach uses a recurrent neural network to assign probabilities to future characters in a candidate guess, based on the previous characters. For example, in generating the string *password*, a neural network might be given *passwor* and conclude that *d* has a high probability of being the next character.

At a high level, in order to generate passwords from a neural network model, the researches enumerate all possible passwords whose probability is above a given threshold, using a hybrid of depth-first and breadth-first search and then sort passwords by their probability. In addition, they evaluate password strength by modeling a guessing attack. Specifically, they calculate a password's guess number or how many guesses it would take an attacker to crack that password, when guessing passwords in descending order of likelihood.

At the implemented neural network password checker, guess number calculation is performed client-side in the browser and neural network computations run in their own thread. Also, for applications such as in a password meter, the authors thought it was desirable to conservatively estimate password strength, meaning that they prefer to underestimate a password's resistance to guessing rather than overestimate it. In order to achieve this, on the client-side model they compute the guess number without respect to capitalization, whereas they don't perform so for the server-side models because those are used to generate candidate password guesses, rather than estimating a guess number. After computing guess numbers, they apply to them a constant scaling factor which acts as a security parameter, to make the model more conservative at the cost of making more errors [9].

Another research based on neural networks for password strength estimation was presented later in 2017 by Ur et al. [13], who extended the concept and proposed a password strength meter that combines two complementary approaches: modeling a principled password-guessing attack using regressive neural networks and combining 21 heuristics that generate data-driven text feedback and try to explain how to improve the password choice, instead of the usual colored bar.

These heuristics search for weak password characteristics, for example the inclusion of common words and phrases, the use of common character substitutions, the placement of digits and uppercase letters in common locations and the inclusion of keyboard patterns. After scoring 30.000 passwords from well-studied data breaches by these heuristics, they ran a regression (linear, logistic, ordinal, multinomial logistic, and Cox Proportional-Hazards regression, depending on the type of the data), comparing these scores to the password's guessability. If a candidate password scores high on a particular heuristic, indicating the presence of a common pattern, text feedback is generated that explains what is wrong with that aspect of the password and how to improve it. For example, a feedback

comment could be "avoid using keyboard patterns" and a suggested improvement for *Mypassword123* might be *My123passwoRzd*.

The authors mention that they generate the suggested improvement as follows. First, once the candidate password is entered, they make one of the following modifications: alter the case of a random letter (lowercase to uppercase, or vice versa), insert a random character in a random position or replace a character at a random position with a randomly chosen character. In addition, if all of the password's digits or symbols are in a common location (e.g., at the end), they move them as a group to a random location. They then verify that this modification still complies with the composition policy requirements and also require that the suggested improved password must be at least 1.5 orders of magnitude harder to guess than the original candidate password [13].

"Heuristics / Lightweight Password Strength Estimation"

In 2017, Guo and Zhang proposed a lightweight password-strength estimation method (LPSE) that measures the strength of a password by comparing the similarity in the structure and the distance between a password given by the user and a standard strong password.

In general, the LPSE algorithm measures the strength of a password, with the cosine-length similarity and the password edit-distance similarity. In order to make this procedure easier, the password is "converted" into a vector $\alpha=(x_1,x_2,x_3,x_4,x_5)$, where x_1,x_2,x_3,x_4,x_5 represent the vector components of digits, lowercase letters, uppercase letters, special characters, and the length of the password respectively. This vector is then compared to a standard strong-password vector. More specifically, the cosine-length similarity can determine the similarity between two password vectors in password structure and password-vector modulus. The edit-distance is the minimum number of operations required to convert a string into another string by inserting, replacing and deleting.

Thus, a given password can be more or less similar to a standard strong password in terms of structure, length, and the number of insertions, substitutions and deletions required to transform it into a standard strong password, which should be low in order for the password to be stronger. To calculate the thresholds of the cosine-length similarity and edit-distance similarity, the authors analyzed various strong and weak passwords that were randomly selected from a large number of leaked password lists. In

addition, in order for the LPSE to identify whether a password contains common, dictionary words, it checks if the password contains common two-character combinations, three-letter combinations, and starting and ending letters. The LPSE algorithm is suitable to be deployed on the client-side [5].

"Heuristics / zxcvbn"

Zxcvbn is a low budget password strength estimator, which is not proposed in the academia but was created and deployed in real life applications by Drobox Inc. Roughly speaking, its standard for measuring a password's strength is the minimum attempts needed to guess it over four guessing attacks. At its core, zxcvbn checks how common a password is according to several sources: common passwords according to three leaked password sets, common names and surnames according to census data, and common words in a frequency count of Wikipedia 1-grams. For example, if a password is the 55th most common entry in one of these ranked lists, zxcvbn estimates it as requiring 55 attempts to be guessed. Zxcvbn is non-probabilistic, instead it heuristically estimates a guessing attack directly and it models passwords as consisting of one or more concatenated patterns. The algorithm assumes that the attacker knows the patterns that make up a password, but not necessarily how many or in which order.

At a high level, zxcvbn consists of three phases: match, estimate and search. At the pattern matching phase, zxcvbn decomposes a given password into patterns and finds a set *S* of overlapping matches. For example, given *Lenovo1111* as input, this phase might return *Lenovo* (password token), *eno* (English "one" backwards), *no* (English), *no* (English "on" backwards), *1111* (repeat pattern), and *1111* (Date pattern, 1/1/2011).

Next, the estimation phase assigns a guess attempt estimation or an "entropy" to each match independently. If *Lenovo* is the 11007th most common password in one of the password dictionaries, it'll be assigned 11007 because an attacker iterating through that dictionary by order of popularity, would need that many guesses before cracking it. In more detail, the final password entropy is calculated as the sum of its constituent patterns' entropy estimates. Patterns as spatial combinations on a keyboard, repeated and common semantic patterns (e.g. dates, years) and natural character sequences are considered weak and given a low entropy. The password is also checked against

dictionaries and is assigned an entropy value based on the average number of trials that an attacker would have to carry out, considering the rank of the password or its subparts in the dictionary. The first and last names as well as the email address of a registering user, are also added to a new dictionary in the algorithm to weaken the strength of a password containing these, which is assigned a very low entropy [7],[8].

The final phase is to search for the sequence of non-overlapping adjacent matches *S* drawn from *S*, such that *S* fully covers the password and minimizes a total guess attempt figure. Following the previous example, the search step would return [*Lenovo* (token), *1111* (repeat)], discarding the date pattern which covers the same substring but requires more guesses than the repeat.

To the extent that zxcvbn is trained on the same or similar password leaks and dictionaries as employed by attackers, Dropbox claims that checking the minimum rank over a series of frequency ranked lists, combined with light pattern matching, is enough to accurately and conservatively predict today's best guessing attacks within the range of an online attack. Zxcvbn is really lightweight and works entirely on the client-side [7].

2.3. Accuracy evaluation and effect of Password Strength Meters

As mentioned, Golla et al. [1] examined, among others, the password strength meters presented above and evaluated them as to their accuracy in measuring password strength. In their work, they conducted a large comparison of different similarity measures and concluded that "weighted Spearman correlation" is best suited to estimate the accuracy of password strength meters. According to weighted Spearman correlation's results, the best performing meters proposed in academia are the ones based on Markov Models, Probabilistic Context Free Grammars (PCFGs) and Recursive Neural Networks. Zxcvbn (heuristics) meter, also performs well.

The effect of password strength meters on password creation has been researched as well.

Ur et al. [16], conducted a 2.931-subject study of password creation in the presence of 14 password meters. They came to the conclusion that meters with a variety of visual representations led users to create longer passwords. However, passwords actually resistant to cracking algorithms were only created in the presence of stringent meters which led participants to spend

more time creating their password and also include more digits, symbols and uppercase letters. In addition, participants who saw more lenient meters, tried to fill the meter and avoided choosing passwords a meter evaluated as "bad" or "poor."

Egelman et al. [17], performed a laboratory experiment to examine whether password strength meters influenced users' password selections when they were forced to change their real passwords and when they were not told that their passwords were the subject of a study. The results of this experiment showed that the presence of meters resulted in significantly stronger passwords. They also performed a follow up field experiment in order to test the creation of a password for an unimportant account (e.g. newsletter accounts). In this scenario, they found that the meters made no observable difference: most participants just reused weak passwords that they used to protect similar low-risk accounts.

While password meters do not yield much improvement in helping users choose passwords for unimportant accounts, they are very commonly deployed in such contexts. They concluded that meters result in stronger passwords when users are forced to change existing passwords on important accounts (e.g. banking accounts).

3. Chapter 3: Targeted password guessing

During a targeted password guessing attack, the adversary tries to guess a specific target's password for a service or an account, by exploiting specific information about them. The adversary might have gained personally identifiable information (PII) about the target (victim), such as names of family members and relatives, usernames, relevant numbers (social security, and phone number), nationalities, addresses (street name, city, state, and zip code), important dates, favorite sports teams and players, etc. as well as possibly some of the target's previous passwords for the same account or from sister accounts. Targeted password guessing is the exact opposite of trawling password guessing, where the adversary aims to crack as many accounts as possible (e.g. by brute force attack), regardless of the owners of the accounts.

3.1. Related work

Wang et al. [11] argue that offline guessing attacks, no matter trawling or targeted ones, only pose a serious threat in a circumstance where the server's password file is leaked, the leakage goes undetected, and the passwords are also improperly hashed and salted. They also note that online guessing is increasingly more damaging and realistic, since it can be launched anytime against the server by anyone using a browser, with the primary constraint being the number of guesses allowed. Hence, they focus their research on online targeted guessing attacks.

As mentioned, targeted online guessing exploits not only weak popular passwords, but also reused passwords as well as passwords that contain personal information. This is a serious security concern, since various personally identifiable information can be leaked through hacking means and data breaches or be publicly available. For instance, over 78.2% of the 668 million Chinese 'netizens' have reportedly suffered PII data leakage. This indicates that the existing password creation policies and password strength meters, such as the ones mentioned above, do not necessarily protect users when PII is included in their passwords. For example, Wang et al. [11], discovered that highly heterogeneous PII becomes components of passwords, and users like to use names, birthdays and their variations. Particularly, 0.75%~1.87% of users employ just their full names as passwords, and 1.00%~5.16% of Chinese users use just their birthdays, while email and user

name prevail, ranging from 0.77% to 5.07% and from 0.54% to 2.34%, respectively. From what is mentioned above, it becomes clear that during a targeted attack there is always the challenge to efficiently choose the best suiting password candidates and create the proper PII combinations from multiple dimensions in order to derive the correct password, while the number of guess attempts allowed by a server's lockout is typically small.

Wang et al. [11], in order to create guessing algorithms that simulate the way adversaries try to guess passwords, divide personal identifiable information into two types: Type-1 and Type-2. Type-1 PII, which may include names and birthdays, can be the building blocks of passwords. Type-2 PII, which for example could include gender, education and nationality, may impact user behavior on password creation but cannot directly be used in passwords. For example, passwords of female users are more concentrated; passwords of users in age ≤ 24 and age ≥ 46 have quite similar length distributions etc.

These password guessing algorithms, can represent four popular attack scenarios based on various kinds of personal information available to the attacker: (i) online guess a user's passwords by exploiting some type-1 PII (ii) online guess a user's password at one service, when given a "sister" password of theirs, leaked from another service (iii) online guess a user's passwords by exploiting one sister password of theirs as well as some PII (iv) online guess a user's password by exploiting one sister password of theirs as well as both type-1 and type-2 PII. Of course, more attack scenarios can occur by creating combinations other than the ones mentioned above. These four algorithms, leverage probabilistic techniques including PCFG, Markov and Bayesian theory.

The authors explicate a preparatory phase that is necessary for these algorithms to work properly, since that's when user or site information are being explored. Consequently, this phase's steps can actually represent the techniques attackers use to gather users' personal information. In more detail, they mention that once the victim has been determined, attackers can deploy profile crawlers or social engineering techniques, collect and study password datasets to find leaked passwords of the victim, and explore site policy to learn the rules a password is subject to. Phishing attacks, are also a great way to gather personal information about a user. After all the information are collected and analyzed, the adversaries can create a personalized prioritized password list and start attempting to login to the victim's account.

Houshmand et al. [12] published their research which was also focused on targeted password guessing. They made the assumption that they had a

single target as well as some obtained personal information about the target besides their password hash. Their goal was to create an attack grammar specifically for generating guesses for each separate target and did not simultaneously target multiple hashes. More specifically, they focus on the dictionary-based probabilistic context-free grammar (PCFG) approach to password cracking that trains on revealed password sets and then uses the learned grammar to generate guesses in optimal probability order. They show how to incorporate personal information about a target into the PCFG password cracking system, to assist in a targeted attack. Based on the fact that two or more probabilistic context-free grammars can be merged, the authors create grammars which contain information about the target-user and each one of these grammars can be matched to a corresponding attack scenario.

At the first scenario, personal information is integrated into the grammar called PI-grammar and also an attack dictionary is created called PI-dictionary that reflects this information and that will be used in password cracking. The PI-grammar used alone, cannot be very useful in password cracking hence they merge this grammar with a general grammar when generating guesses. At the second scenario, it is shown how to build and use a grammar for a targeted attack if information about previous passwords of the victim is known. Old passwords can contain vital information such as important numbers, dates or family member names and the goal here is to define a grammar that can generate guesses similar to an old password. At the third scenario, it is assumed that the attacker has much more information about the victim. The authors focus on the knowledge about the victim's password habits and assume that the attacker has access to at least two similar subsequent old passwords. To determine the changes between two passwords, they implemented a function that finds the minimum edit distance.

Once the attacker creates any or all of the above grammars based on the kind of information available, the authors suggest merging all the grammars together with a more comprehensive general grammar that would be used for password cracking when no personal information was known. By assigning appropriate weights to each grammar, they try to balance the generated guesses so that guesses with personal information will generally be generated earlier with higher probability values [12].

A similar paper that tried to integrate personal information into PCFGs, is the one by Li et al. [14]. In their work, the authors extended the approach of Weir et al. to develop a system they called Personal-PCFG.

Dürmuth et al. [15] on the other hand, didn't base their research on PCFGs but rather explored how to use personal information to improve password cracking attacks, using the Markov model approach by Narayanan and Shmatikov. They developed a system called OMEN+ that could integrate personal information with normal training on revealed passwords. The basic disadvantage of the Narayanan et al. Markov approach, was not outputting passwords in order of decreasing probability. That is because it is computationally difficult to generate passwords in highest probability order, since the training typically uses the learned probability of n-grams to determine the probability of a password string. The algorithm they developed, enumerated passwords with (approximately) decreasing probabilities.

On a high level, the algorithm discretizes all probabilities into a number of bins, and iterates over all those bins in order of decreasing likelihood. For each bin, it finds all passwords that match the probability associated with this bin and outputs them. In other words, they created various discretized probability levels that were assigned to the bins in which strings were placed. The strings in these bins were then tried in the highest probability order of the buckets. The authors next explored the use of personal information (but did not explore using older or cross-site passwords).

To assess whether social information could be exploited to improve password cracking, they quantified the correlation between passwords and personal information by using two different similarity metrics to capture different aspects of the potential overlap. *Longest common substring (LCSS):* The LCSS of two strings is the longest string that is a substring of both strings. *(Modified) Jaccard similarity (JS):* The Jaccard similarity index compares similarity of two sets not strings. Because appending unrelated information to the personal information rapidly decreases the JS value, they use a modified JS.

OMEN+ tries to first determine if a password set containing personal information overlaps at any point with the password containing the personal information. If so, the corresponding probabilities of the overlapping n-grams are boosted based on a computed boosting parameter, so as to increase the probability of passwords related to them, and thus improve OMEN+ performance.

3.2. Nationality-based targeted password guessing

In the following chapters, we will briefly analyze the different habits of non-English users in terms of how they create their passwords. We will also examine how two widely used Password Strength Meters handle non-English passwords, especially focusing on Greeklish passwords. More specifically, we will measure the average scores assigned to the Greeklish passwords before and after training these Password Strength Meters with a Greeklish training set, to see if their strength is being over-estimated or not.

In addition, we will analyze how various Password Cracking Tools handle passwords containing non-ASCII characters, e.g. letters from the Greek alphabet and finally, simulate a targeted password guessing attack in which the attacker is based on the nationality of the users.

4. Chapter 4: Password Strength Meters and non-English passwords

Users from different cultural/linguistic backgrounds may prefer different patterns to create their passwords and thus, knowledge on English passwords cannot always help to guess and correctly assess passwords coming from other languages. In fact, various researches have concluded that users from different demographical groups tend to develop different password habits. This means that they show differences in how they form their passwords or how they incorporate their PII into their passwords and the extent to which they do so [19].

The main reason different password habits occur, is because people from other cultures think differently and hence choose passwords differently. For example, Chinese-speaking users often use their mobile phone numbers or certain dates (perhaps birthdays) in their passwords, something that English-speaking users don't do as often. Another reason could be the fact that non-English speaking users often choose words from their mother tongue and then use English language to phonetically transcribe and type it, for example *woaini* in Chinese pinyin (Romanized system of Chinese characters) stands for *I love you* [18]. Spelling mistakes could be another important reason why different password habits occur as they tend to change the word, sometimes significantly.

Even in the cases where a person coming from a different culture creates their password in the same way an English-language user would, for instance they incorporate their PII into their passwords, the results might still get affected. For example, some Arabic names can be spelled in several ways, thus creating variations of the same word.

The preceding observations can be quite concerning for password security, considering the fact that existing password strength meters are, by a large percentage, based on common base words and patterns observed from previous data leaks of mainly English-speaking users. That is because, what may seem like a strong password based on English-language assumptions, could actually be quite weak and easy to guess from a foreign language perspective. Following the previous example, the popular Chinese password *woaini1314*, is currently rated "strong" by well-known password strength meters but this estimation is biased, as speakers of Mandarin Chinese could easily guess it [18]. As a result, existing PSMs cannot provide accurate results on password strength estimation.

4.1. Related work

Interest in this field has grown in the past years, but research is still in early stages for us to obtain satisfactory results. Most related research, examines the differences between English and Chinese passwords, which is not surprising given that Chinese people make up more than 20 percent of all Internet users worldwide.

Li et al. [20] conducted a large-scale analysis on Chinese passwords and studied the differences between passwords from Chinese and English-speaking users, taking advantage of over 100 million leaked and publicly available passwords from Chinese and international websites. They also improved a PCFG-based password guessing method, by inserting Pinyins into the attack dictionary and observed composition rules into the guessing rule set. This experiment, showed that the efficiency of password guessing increased by 34%.

Wang et al. [21] also conducted an extended, empirical analysis of 73.1 million Chinese web passwords, in comparison with 33.2 million English counterparts. They discovered a number of interesting structural and semantic characteristics in Chinese passwords, and also employed two PCFG-based and Markov-based password attacking algorithms to further evaluate the strength of Chinese web passwords. They concluded that Chinese passwords are weaker against online guessing attacks but stronger against offline guessing attacks than English passwords.

AlSabah et al. [19] on the other hand, conducted a study based not on Chinese users' passwords, but on metadata and passwords coming from a Middle Eastern bank's data leak. They analyzed passwords created by groups such as Arab, Filipino, Indian, and Pakistani, which are not traditionally present in similar researches. They illustrated the differences in how users from different cultural/linguistic backgrounds create passwords, how they incorporate their PII in their passwords and also studied the guessability of the dataset based on two attacker models. They concluded that the state-of-the-art password strength estimator: zxcvbn, inflates the strength of passwords created by users from non-English speaking backgrounds, when compared against their English counterparts.

4.2. Greek and Greeklish passwords

Greeklish, a combination of the words "Greek" and "English", is the Greek language transliterated using the Latin alphabet; est. words in Greeklish derive from the exact replacement of each Greek letter with its Latin respective. For example the Greek word "καλημέρα" which means "good morning", would be "kalimera" in Greeklish after the following matches: κ=k, α=a, λ=l, η=i (or h), μ=m, ε=e and ρ=r. Greeklish is mostly used by Greeks when surfing or communicating over the Internet, but is also widely used in password creation.

To our knowledge, there is no recent study that examines the differences between English and Greeklish passwords or the strength of Greeklish passwords against any of the aforementioned password strength meters. Two studies focused on Greek users' passwords in general, are mentioned next.

Voyiatzis et al. [22], conducted an empirical study on the strength of web passwords in Greece. They analyzed cleartext as well as encrypted passwords for almost 19.000 accounts, while mostly focusing on characteristics such as password length, password patterns and if Greek alphabet characters are being used in passwords. They concluded that for Greek users, the average password length is slightly less than 7 characters and 99.6% of the passwords consists of Latin characters and digits only. Furthermore, they encourage users to choose passwords based on their native-language alphabet, given that an extended character set as password input is beneficial from a security point of view.

Violettas et al. [23], tried to describe the password habits of Greek web users too. For this purpose, they constructed an online questionnaire in order to gather information about the length of the password users use today, in how many web sites they use the same password, whether they use personal data while creating their passwords, if they ever used trivial words as a password, if they ever revealed their password to a relative and if their password was ever exposed to an attacker. They concluded that while a big percentage of the users answering the questionnaire had never used an "easy" password, there was still a majority of users who were just using three, two or even one password for all their online services. They also ended up with a list containing the 100 most common Greek passwords, which users should absolutely avoid nowadays.

It is therefore becoming clear that more systematic research is needed, which will uncover the actual strength of Greek users' passwords against modern password strength meters, as well as if they would be secure during a password cracking attack where the password cracking tools have been trained with a Greek wordlist, and which in overall will contribute in better understanding if and how password metrics are affected from different languages.

4.3. Creating a Greeklish password list

The first crucial step, was to find at least one leaked Greeklish password dataset that would be the key element of our study, but we faced difficulties doing so. To our surprise, during the time period of our study, we were not able to find such password dumps hosted publicly, that would either come from a Greek website or public service system etc. In the study of Violettas et al. [23], even though the three datasets consisted of passwords from real web sites in Greece, two of them were made available specifically for the study and the third one was posted publicly only for a short period of time in the past.

Another alternative, would be to use a dataset made up of many different individual data breaches from thousands of different sources, hosted at https://haveibeenpwned.com that included over 2.5 billion passwords. Unfortunately, each password in that list was being represented as either a SHA-1 or an NTLM hash to protect its original value, since many of them contained personally identifiable information, and trying to crack them would be a time-consuming process that went beyond the scope of our study.

Another peculiarity we had to handle, was that we specifically needed passwords comprised of Greeklish words and thus acquiring a password dump that came from a Greek site or included accounts from Greek users, still wasn't going to be enough. In other words, knowing that the password e.g. "123abcd" belonged to a Greek user did not fulfill the purpose of our study, since it wasn't about analyzing the password creation habits of Greek users but rather about investigating how modern PSMs react to Greeklish passwords.

4.4. Datasets gathering

In order to be able to continue with our research, we had to create a Greeklish dataset of our own. Our initial thought, was to use leaked-password lists coming from websites that are predominantly used all around the world, search if they contained any Greeklish passwords and finally collect them. We decided to go with leaked password lists from three big websites, LinkedIn, Dropbox and Yahoo which are hosted publicly online, at https://hashes.org/.

We also needed a Greeklish dictionary [24], i.e. the Greeklish version of words coming from an actual Greek dictionary. The dictionary would be used as an "extractor" for the Greeklish passwords, i.e. we would search for Greeklish passwords in the leaked password lists against this dictionary, since this was not a procedure that could be done manually due to the large size of the lists. One very interesting fact about the wordlist we finally chose, was that it contained different Greeklish permutations. For example, the word "ψυχή" (soul) is included in the following variations: psyxh, psixh and psixi, which are the most usual ways in which someone can write that word in Greeklish.

Another thing we know, is that a portion of the users tend to include their first and/or last names in their passwords. Hence, in order to be able to also extract such passwords, we searched for the top 10 most common first and last, female and male names in Greece and we added their different permutations in the Greeklish dictionary mentioned above. This way, we could now easily observe how PSMs score passwords including names such as "Kwstas", which is widely used in Greece but not in other countries.

4.5. Creation of a Greeklish password dataset

After having collected all the necessary parts, we were ready to start creating our own Greeklish password dataset. In order to achieve this, we created a script that would take two files as input: the first one would be one of the leaked-password lists we had chosen and the second one would be the Greeklish dictionary. It then would compare each word in the Greeklish dictionary against all the words in the leaked-password list so as to find a matching string or substring between them. In other words, the script would look for passwords that either exist as whole words in the Greeklish dictionary, or contain a word from the Greeklish dictionary as part of them. Finally, if it actually managed to find such a match, it would print the passwords in a third file.

It is also worth mentioning that the Greeklish dictionary contained all kinds of words, even single character ones, so we had to remove all the words that had less than five characters before the beginning of the comparison procedure, in order to produce as best results as possible. Otherwise, all the words from the leaked password lists would end up in our Greeklish password dataset and not just the ones containing Greek words. At the end of this process, we were left with three new files containing our Greeklish password datasets.

The final version of our datasets was as follows. The initial LinkedIn list contained 60.636.305 cracked passwords while the Greeklish LinkedIn list contained 3.413.326 cracked passwords, the initial Dropbox list contained 9.869.036 cracked passwords while the Greeklish Dropbox list contained 413.514 cracked passwords and the initial Yahoo list contained 5.951.732 cracked passwords while the Greeklish Yahoo list contained 308.958 cracked passwords.

5. Chapter 5: Greeklish password dataset analysis with Password Strength Meters

At this point, we were finally able to start analyzing our three datasets against password strength meters, in order to examine their results when measuring the strength of Greeklish passwords before and after they are trained with a Greeklish wordlist. We decided to use two password strength meters; Dropbox's zxcvbn which was also described in Chapter 2, and pcfg_cracker which is based on a probabilistic context-free grammar (PCFG).

5.1. Analysis with zxcvbn

For the analysis using zxcvbn password strength meter, we followed the same tactic as AlSabah et al. [19], we described above. More specifically, we selected the Java version of the zxcvbn library which is opensource and is hosted publicly online at https://github.com/nulab/zxcvbn4j.

In order to analyze the datasets, we used the *zxcvbn.measure* function that would take as input our three Greeklish datasets, from LinkedIn, Yahoo and Dropbox. The function would then read each dataset line-by-line, measure the strength of the password using zxcvbn library, and print its output in a third file. Zxcvbn assigns a score to the password from a range of 0-4, where 0 stands for a too guessable password (guesses < $3^{\wedge 10}$) while 4 stands for a very un-guessable password (guesses > $10^{\wedge 10}$). The Java version also provides the option to print other useful information about the password besides its score, such as feedback on how to make the password stronger, calculation time, number of guesses needed, etc.

On the first password-strength-scoring session, we had our Greeklish passwords scored using the default zxcvbn implementation which is based solely on English wordlists for its training, and we would expect to overestimate their scores. The zxcvbn Java version default training sets were:

1. english_wikipedia
2. female_names
3. male_names
4. surnames
5. passwords
6. us_tv_and_films

The female_names and male_names training list, contained mostly English and Latin names, as for the rest of the lists it is obvious that they contained English words as well. The scoring process took about 2 minutes for the Dropbox and Yahoo password lists - that were smaller in size - and about 12 minutes for the LinkedIn password list - that contained the largest number of leaked passwords -.

For the second password-strength-scoring session, it was time to evaluate the passwords against a "Greeklish-trained" version of the zxcvbn password strength meter. As previously mentioned, we did not have a Greeklish password set at our disposal, so we had to carry out the training using a different dataset. This training set was the Greeklish dictionary we had also used during the password extraction phase, analyzed in Chapter 4. Moreover, in order for the training set to be more complete, we combined the Greeklish dictionary with a second list of about 3.5 million words including Greeklish first and last names provided here [26]. The creator of this list aimed to simulate a set of real passwords, since they also were not able to find a similar one for Greek users. The list contains first and last names combined with dates and special characters, l33t transformations and reversed words. Thus, the final training set contained about 7 million words.

In order to carry out the training, we first needed to create a Java List object that would store the training set data. Then we could run the scoring session again, only this time we needed to use a different implementation of the *zxcvbn.measure* function that took two inputs; the leaked password list and the training set. Therefore this time, the strength of the Greeklish passwords was measured against our training set. The scoring process using a training set lasted significantly longer than before, i.e. about 12 hours for the Dropbox and Yahoo password lists and more than 24 hours for the LinkedIn password list.

The results of the analysis are described in Chapter 6.

5.2. Analysis with PCFG

For the analysis using Probabilistic Context Free Grammars (PCFG), we made use of *pcfg_cracker* tool [25]. Pcfg_cracker is an open-source implementation of a PCFG, whose documentation is based on academic papers. It also is an active project, meaning that new versions are being developed and released from time to time.

This PCFG model can be trained in order to generate a ruleset which contains various different parts of the passwords it identified, along with their associated probabilities. Its main purpose, is to use machine learning to identify users' password creation habits by training on disclosed password lists.

The one feature that interested us most for our research though was 'Password Strength Scoring', which estimates the probability of a password being generated by a ruleset like the one mentioned above or, in other words, the probability of it being guessed by such a password cracker. More specifically, the password strength scoring feature of pcfg_cracker takes two inputs; the list of passwords to score along with a previously generated ruleset, and estimates the probability of the password according to the ruleset. If the probability is assigned a value of 0, it means that the password will never be generated using this particular ruleset. In other words, higher probabilities indicate that pcfg_cracker considers the passwords easier to be guessed.

In order to see if and how the results of the PCFG password strength meter would differ, we measured the strength of our three, leaked, Greeklish password datasets using four different rulesets, as described in the following paragraphs.

Pcfg_cracker was originally trained by its developers with a one million password subset of the RockYou dataset - which is a fairly small number for a training set - and most likely, the strength guessing results for any set of passwords, even English ones, would be misleading. Nevertheless, for reasons of completeness of our research we decided to run the first password-strength-guessing session of the Greeklish passwords, against the default ruleset.

In order to derive more representative results, it was essential to train pcfg_cracker with a larger password set. For this reason we used a list that contained over eleven million passwords that had been cracked in the period between January and April 2020, and was hosted at www.hashes.org. The second password-strength-guessing session of the Greeklish passwords, was against this ruleset which we named "2020 ruleset".

Subsequently, it was time to train the PCFG model with the Greeklish dictionary combined with the Greeklish words list we had previously used during the zxcvbn analysis, and see if the probabilities of our Greeklish passwords being generated by the ruleset, would increase. During the

training we also changed the value of the coverage. In short, the coverage flag represents how much you trust the training set to match the target passwords. A higher coverage value, means to use less intelligent brute force generation thus by setting coverage to 1, no brute force will be performed. If you set coverage to 0, it will only generate guesses using Markov-based attacks. Default coverage value is 0.6, which means it expects a 60% chance for the target password's base-words to be found in the ruleset.

In our case we set the coverage value to 1, since we knew that we would use the ruleset against a Greeklish password list so we expected a 100% chance that their base-words would be found in the Greeklish ruleset. Similarly, during a password cracking attack where the attacker aims to crack Greeklish passwords only, they can train the PCFG model accordingly and set the coverage value to 1 in order to increase their chances of cracking the passwords. The third password-strength-guessing session of the Greeklish passwords, was against this ruleset which we named "Greeklish ruleset".

Having created the third ruleset, we came to realize that even though we had correctly chosen to train pcfg_cracker with the Greeklish dictionary, the results might still not be satisfying enough. First of all, the "Greeklish ruleset" was too small in size compared to the 32 million passwords that was suggested by the developers of pcfg_cracker tool as an adequate training set size. Second, we considered that the "Greeklish ruleset" did not contain a sufficient amount of data that could simulate more realistic passwords, such as enough combinations of alphanumeric words and symbols. Trying to overcome these obstacles, we decided to combine the second and third ruleset and thus create one big ruleset with about 18 million words and various different alphanumeric combinations, hoping that it would be closer to our target. The fourth password-strength-guessing session of the Greeklish passwords, was against this ruleset which we named "full ruleset".

6. Chapter 6: Results of the Greeklish password dataset analysis

The analysis we carried out in the previous chapter both using zxcvbn and pcfg_cracker password strength meters, gave the results that are described in the following paragraphs.

6.1. Zxcvbn analysis results

As mentioned above, the first password-strength-scoring session was ran against the default zxcvbn implementation, while the second password-strength-scoring session was ran against the zxcvbn that had been trained with the Greeklish dictionary and the Greeklish names list. More specifically, in order to determine whether the training set changed the way zxcvbn scored our passwords, we calculated the average scores before and after training.

Before training, the average score for Dropbox, Yahoo and LinkedIn password sets was around 3, which translates as 'strong', while after training their average score was around 2.25, which translates as 'good', meaning that it was decreased by over 0.6 points. The scores are displayed in more detail in Table 1.

	Dropbox	Yahoo	LinkedIn
Before Training	2.89	3.01	3.08
After Training	2.12	2.28	2.38

Table 1: zxcvbn password scores before and after training

Though the before and after training scores vary, in overall they did not decrease as much as we hoped for. This is mostly due to the fact that all three datasets also included non-Greeklish passwords which were however not matched as such during the password extraction phase, because they contained a Greeklish substring. This resulted in them being scored leniently and hence dropping the average.

Taking a closer look at the before and after training scores of some randomly picked Greeklish passwords however, one can easily conclude that zxcvbn actually inflates the strength of non-English passwords when using the

default training set. The passwords along with their scores are presented in detail in Table 2.

Passwords	Before Training	After Training
Antistrateuomai1 (LinkedIn)	4	2
katapliktikoteros (LinkedIn)	4	2
tsakmakopetra.24 (LinkedIn)	4	3
Diastrwmatwsh1 (LinkedIn)	4	2
diasundeseis10 (LinkedIn)	4	2
autokollito81 (LinkedIn)	4	3
trelikatsarida (LinkedIn)	4	3
xeimonas (LinkedIn)	2	1
migdalia560 (Dropbox)	4	2
kalimera1961 (Dropbox)	4	2
pontikaki (Dropbox)	3	1
oneiropagida (Dropbox)	4	4
portokalada40 (Dropbox)	4	2
giagiachristina1 (Dropbox)	4	3
ergasiakaixara (Yahoo)	4	4
ergolaboi18 (Yahoo)	4	2
thalassa27! (Yahoo)	4	3
Kouloura1 (Yahoo)	3	2
eimaidaskala (Yahoo)	4	3
kalampoki22 (Yahoo)	4	2
	Average	
	4	2.4

Table 2: zxcvbn scores for Greeklish passwords before and after training

We notice that the scores differ by at least one point, while in most cases they differ by two points. Less often, the before and after scores are equal, meaning that probably a more complex training set is needed in order for zxcvbn to be able to produce more valid results.

The before training average is 4 which is the highest score that can be given from zxcvbn and translates to ‘very strong’ and on the other hand, the after training average is 2.4 which translates to ‘good’.

6.2. Pcfg_cracker analysis results

As mentioned in Chapter 5, we ran four password-strength-scoring sessions using the default ruleset, the 2020 ruleset, the Greeklish ruleset and the full ruleset accordingly. In order to determine whether the training set changed the way pcfg_cracker scored our passwords, we calculated the average scores before and after training, just like we did for zxcvbn.

The average password guessing score using the default ruleset was 0.56% for the Dropbox password list, 0.46% for the Yahoo password list and 0.070% for the LinkedIn password list. Surprisingly enough, the default ruleset gave the highest guessing probabilities among all rulesets which meant that, at least seemingly, the meter could guess more passwords using this simple ruleset.

The average password guessing score using the 2020 ruleset was 0. 0118% for the Dropbox password list, 0.11% for the Yahoo password list and 0.021% for the LinkedIn password list.

The average password guessing score using the Greeklish ruleset was 0. 043% for the Dropbox password list, 0.046% for the Yahoo password list and 0.018% for the LinkedIn password list. As mentioned previously, the probabilities of the Greeklish ruleset are low, because it contained a lower percentage of complex alphanumeric data, concatenated words, etc. that could simulate actual passwords. Since pcfg_cracker is solely based on the ruleset, it was not able to efficiently calculate password guessing probabilities for the passwords in our three leaked lists.

The average password guessing score using the full ruleset was 0. 070% for the Dropbox password list, 0.065% for the Yahoo password list and 0.015% for the LinkedIn password list. The scores are displayed in more detail in Table 3.

	Dropbox	Yahoo	LinkedIn
default ruleset	0.56 %	0.46 %	0.070 %
2020 ruleset	0.0118 %	0.11 %	0.021 %
Greeklish ruleset	0.043 %	0.046 %	0.018 %
full ruleset	0.070 %	0.065 %	0.015 %

Table 3: Pcfg_cracker password scores using 4 different rulesets

The full ruleset managed to increase the guessing probabilities both for the Dropbox and the Yahoo password lists, but did not do so for the LinkedIn password list. Nevertheless, they were still not as high as the ones coming from the default ruleset; it seemed that, for some reason, pcfg_cracker perceived Greeklish passwords as more difficult to crack when using a Greeklish ruleset. When we took a closer look at the results, we realized that this was happening because there were many 0% values among the results of the default ruleset scoring session, and hence did not drop the average. In other words, the scores were high simply because there were many

passwords that pcfg_cracker would have never guessed while using the default ruleset.

For this reason, we decided to run the scoring sessions once more, but this time we would also count the number of non-guessed passwords for each of the four rulesets. As expected, the default ruleset had a high total of non-guessed passwords. The Greeklish ruleset also had a high total, for the same reasons as those mentioned above. The 2020 ruleset and the full ruleset had much lower totals and therefore, even though their guessing probabilities were lower, in reality they were able to guess a much larger number of Greeklish passwords.

	Dropbox	Yahoo	LinkedIn
default ruleset	203.895	244.535	2.322.267
2020 ruleset	93.701	101.843	1.088.840
Greeklish ruleset	295.730	393.223	3.311.007
full ruleset	88.633	94.983	1.028.807
	Total		
	308.957	413.512	3.413.325

Table 4: Non-guessed passwords by pcfg_cracker using 4 different rulesets

We can gain take a closer look to some passwords presented in Table 5 and examine the guessing probabilities pcfg_cracker appended, using the four different rulesets.

Passwords	default ruleset	2020 ruleset	Greeklish ruleset	full ruleset
Antistrateuomai1 (LinkedIn)	0	0	0.0000039	0.0000015
katapliktikoteros (LinkedIn)	0	0	0.014	0.031
tsakmakopetra.24 (LinkedIn)	0	0.000008	0	0.000003
Diastrwmatwsh1 (LinkedIn)	0	0	0.0000065	0.000035
diasundeseis10 (LinkedIn)	0	0.000003	0	0.000008
autokollito81 (LinkedIn)	0	0	0	0.00007
trelikatsarida (LinkedIn)	0	0	0	0
xeimonas (LinkedIn)	0	0	0.0029	0.000028
migdalia560 (Dropbox)	0.002	0.02	0	0.00048
kalimera1961 (Dropbox)	0.00024	0.00067	0	0.000096
pontikaki (Dropbox)	0	0.00017	0.02	0.000029
oneiropagida (Dropbox)	0	0	0	0.0000022
portokalada40 (Dropbox)	0	0	0	0.0000055
giagiachristina1 (Dropbox)	0	0.000033	0	0.0000033
ergasiakaixara (Yahoo)	0	0	0	0
ergolaboi18 (Yahoo)	0	0	0	0.00026
thalassa27! (Yahoo)	0.000045	0.000022	0	0.00003
Kouloura1 (Yahoo)	0	0.0000063	0.00007	0.0000015

eimaidaskala (Yahoo)	0	0.000002	0	0.0000016
kalampoki22 (Yahoo)	0	0.000002	0	0.000023

Table 5: pcfg_cracker scores for Greeklish passwords using 4 different rulesets

Based on the scores presented in Table 5, we can easily conclude that the default ruleset would be able to guess a few of the passwords which were included in our three lists, even with a higher precision in some cases. However, it would still not be able to guess most of the passwords containing Greeklish substrings, which is the main goal at this stage of our study.

6.3. Conclusions on the results

The results of both the zxcvbn and the pcfg_cracker analysis, confirmed our initial hypothesis that most modern Password Strength Meters tend to overestimate the strength of non-English passwords, in our case Greeklish passwords, because their training is mostly based on leaked password set of English speaking users. In addition, we proved that when trained with a suitable set, e.g. a Greeklish wordlist, the password scores will drop significantly. Hence, the belief that non-English passwords are actually stronger is not necessarily true and they might still be guessed by a properly prepared adversary.

We believe that we would be able to draw an even clearer picture, especially for the analysis using pcfg_cracker, if we had a better leaked password list as well as a better training set at our disposal. For example, an actual password set leaked from Greek websites or a list of passwords from Greek users that would mostly contain various Greeklish words, duplicates, etc. would be the perfect alternative. That way, we would be able to easily analyze the password creation habits of Greek users as well, besides password strength.

7. Chapter 7: Non-ASCII characters in passwords

In the previous chapters, we mentioned that users from different cultural backgrounds tend to develop different password creation habits, and often choose words from their mother tongue which they phonetically transcribe and type in English. We also examined how various Password Strength Meters handle passwords that are comprised of non-English words, and concluded that in most cases they overestimate their strength, because the majority of PSMs are trained with common English base words or patterns, and passwords of mainly English-speaking users, collected from previous data leaks.

Besides non-English words though, passwords can contain non-English characters as well. For example, characters in the Greek, Russian or Chinese alphabets as well as characters in the French or Spanish alphabets etc. that have special accents, are considered non-Latin/non-ASCII[1]. Hence, it would be interesting to extend our previous research and further examine how Password Cracking Tools behave with such passwords, but also how practical it is to use non-ASCII characters in passwords, if today's systems are able to support them, and most importantly if they provide any additional level of security against password cracking attacks compared to passwords containing Latin characters.

7.1. Practicality of non-ASCII characters usage in passwords

One important aspect that should be taken into account before a person decides to include non-ASCII characters in their password, is if this password will later be practical to use.

First of all in order for someone to be able to type non-ASCII characters, it is necessary to have an operating system and a physical keyboard that support such inputs. There are various circumstances outside the user's control however, that might get in the way of entering their password, as it is not guaranteed that anywhere it is likely needed to type that password in, there will be access to those non-ASCII characters from the keyboard.

Such a case would be to have to use a different terminal like a public computer or a new smartphone, etc. whose keyboard comes with only one

[1] Since ASCII is the original character encoding schema used for Latin/English characters, we will refer to all non-Latin characters as non-ASCII.

language installed - most probably English -. In a situation like this, entering non-ASCII characters might require either typing long sequences of keys which represent the individual strokes of a character (e.g. in Chinese) or using a graphical interface in addition to the keyboard [27]. As these methods are either too complicated to learn or vulnerable to crowd-surfing, it becomes obvious that if the password contains characters outside of the usual ASCII set, then there is a high chance that users will not be able to conveniently login when ran into similar situations.

7.2. System support of non-ASCII characters

It is not unlikely that some systems may not support or allow non-ASCII characters, since their backend, database, etc. would have to be properly regulated for such occasions, otherwise unexpected things will possibly happen. The reason behind this, is that we cannot always know for sure what is actually received by the backend when one presses a non-ASCII character, since its underlying byte sequence can be different each time, depending on factors such as the operating system's setup or the keyboard, etc.

Passwords on the web for instance, are typically submitted via an HTML form which requests a particular encoding (e.g. using the accept-charset attribute). As mentioned in some of the following paragraphs however, it is common for the encodings requested by the front-end to differ from the encodings actually used by the server and hence conversion is required [27]. This is not always an easy process, as character à for example, is encoded in Extended ASCII as *'(hex)85'*, in ISO Latin-1 as *'(hex)E0'*, in Unicode it is *'U+00E0'*, and in UTF-8 *'(hex)C3(hex)A0'*. Consequently the conversion procedure is prone to mistakes, unless it is performed with caution.

Another implication arises for non-ASCII passwords when Percent Encoding (or URL Encoding) is taken into account. URLs can only be sent over the Internet using the ASCII character-set, but they often contain characters outside the ASCII set as well, so Percent Encoding is used to convert any "unsafe" character from the URL into a valid ASCII format. Specifically, as Bonneau and Xu mention in [27], *"any byte value outside a limited subset of ASCII must be converted to percent sign % followed by the byte value in hexadecimal"*.

Percent Encoding technique is also applied on passwords, since browsers cannot directly submit an encoded password. Thus replacing passwords with a numeric entity reference and percent encoding them, can result in them

being expanded into a large number of bytes per character. The single Chinese character 爱 for instance, will be expanded as %26%2329233%3B in ISO 8859-1. Now if at a later time the server attempts to enforce password length restrictions, it can lead to obvious errors. For example IMDB, enforces a maximum password length of 64 bytes, restricting users to only 4 characters when not using ISO-8859-1 encoding. The case where a not Unicode-aware system may interpret one multi-byte Unicode character as multiple single byte characters, falls into the same category since individual UTF-8 characters can be escaped to as many as 14 bytes each.

Furthermore, there are sites surprisingly including Google, Microsoft Live, Yahoo! and Amazon among others, which have an explicit policy omitting non-ASCII characters in passwords [27].

Last but not least, there are 20 or 30 year old legacy systems out there still being used, and they only support a small subset of ASCII.

Thus, it becomes clear that when a user decides to use passwords that are entirely comprised of, or contain some non-ASCII characters, they should keep in mind the following restrictions. Everywhere they will be entering their password should allow them to switch languages, the system should be designed as to correctly accept their input and, of course, the encoding mechanisms should be implemented consistently throughout the entire system - for each specific system they might need to enter their password to - otherwise they will most probably get locked out of their accounts.

7.3. Security of non-ASCII passwords

When examining the security of non-ASCII passwords, one should take two important aspects into consideration. First of all, if systems can properly handle such passwords security-wise and second, if using non-ASCII characters actually enhances password strength in real life.

7.3.1. System handling of non-ASCII passwords security-wise

Bonneau and Xu [27] conducted a research where, among other things, they examined the different ways various sites tend to mistreat passwords with non-ASCII/non-Latin characters. More specifically, they examined 12 primarily English-language and 12 primarily Chinese-language sites using a mix of encodings and overviewed several bugs resulting from servers not

handling character encoding issues, most often due to poorly designed back-end code.

One of the cases mentioned in their research, deals with Unicode code point truncation. In character encoding terminology, we refer to characters by their Universal Character Set/Unicode code point and during Unicode code point truncation, code points are truncated to particular byte lengths while inflicting the least amount of data corruption possible. What Bonneau and Xu discovered, was that although two Chinese websites expressly prohibited non-ASCII passwords, they did not actually impose any restrictions on the submitted passwords. Instead, they masked each character's code point returned by the function *CharCodeAt()* with 0xFF, which resulted in all the Unicode characters a, Ł, c (Cyrillic c), ⁵, 屁 to be considered equivalent because they all shared the same code point, ≡ 41.

Other truncation-related cases are those of a Chinese and a Taiwanese site, which limited their passwords lengths to 4–8 UTF-8 characters. Nevertheless, on their back-end servers the function *DES-crypt()*, based on the obsolete DES encryption method, was being used. In short, DES-crypt() works by discarding all the input bytes besides the first 8 and compressing them into a 56-bit key by truncating each byte to 7 bits. For example, when submitting a password where the first three characters encoded in UTF-8 took up 9 bytes, DES-crypt() would simply ignore the remaining ones. As a result, an adversary would only need to correctly guess the first 2 bytes of the last character right, and in the worst case scenario, they may only need to guess two characters if the user happened to choose characters with a 4-byte UTF-8 encoding, which is common in the Chinese alphabet.

One more bug that the authors observed to occur when DES-crypt() processes passwords with non-ASCII characters is called string termination bug and it is caused by the way DES-crypt() copies password to its internal buffer. Passwords are truncated upon seeing either of the bytes 0x00 or 0x80, the latter of which can appear in the middle of non-ASCII passwords, like character `A for example, where in UTF-8 is represented by the bytes 0xC380. This means that the hashes of the passwords *`A* and *`Apassword* will be identical, since anything after character `A will be ignored.

The cases mentioned in the last three paragraphs, represent severe security flaws which can easily be taken advantage of by an attacker against users choosing non-ASCII passwords, as different submitted passwords will end up being mistakenly accepted by the system. Besides that, they can also create confusion to users logging-in to their accounts.

Lastly, they mentioned that although two websites claimed that they allowed their users to use non-ASCII passwords, they later mistakenly converted those passwords on the server-side to ASCII encoding and replaced all non-ASCII characters with the '?' character. This bug made password guessing attacks overly easy, since the adversary would simply need to replace any potential characters outside the server's encoding with the default '?'.

To sum up what was mentioned so far in this chapter, as we do not always have insight into how a website is sanitizing passwords, users could easily end up in a situation where although they have submitted some long Unicode,
non-ASCII, etc. password, after sanitation it ends up being stored as e.g. a hash of very few ASCII printable characters it contained - if it contained any - or being truncated to only a fraction of the originally submitted non-ASCII characters. Consequently, users will be rest assured thinking they have very good security which is far from the truth, since they will still be able to login with their non-ASCII password, but so will anyone else that can brute-force a few ASCII characters or will just type in enough Unicode characters that will end up being sanitized anyway.

7.3.2. Security levels of non-ASCII passwords

For a long time, password composition policies claimed that in order for a password to be considered secure, it should contain complex combinations of uppercase and lowercase letters, numbers and symbols. *Comprehensive8* is an example of such password composition policy, were passwords must include at least 8 characters, including a lowercase English letter, an uppercase English letter, a digit, and a symbol [28]. The last few years, researchers tend to be more in favor of policies requiring longer passwords with fewer requirements, as experiments showed they can be more usable, easier to recall and in various circumstances more secure than policies like *Comprehensive8*.

As we previously noted in Chapter 2, during various password cracking experiments, passwords created using *Basic16* composition policy (passwords should contain at least sixteen characters with no further restrictions), were cracked significantly less than any other condition, at one trillion guesses. Specifically, all conditions except one were proven stronger than *Comprehensive8*. NIST also estimates that such sixteen-character passwords, have up to 48 bits of entropy against passwords of *Comprehensive8* family that have up to 30 bits of entropy [28].

We have seen so far, that including characters outside the typical ASCII space in to passwords is not the best option in terms of usability and system support. We can take all the aforementioned findings into consideration, when trying to determine whether non-ASCII characters actually make passwords safe as well.

New data show that length beats a larger character space size every time, when it comes to password entropy. In addition, as mentioned in Chapter 7.3.1, poorly implemented systems may truncate non-ASCII passwords to only a few characters. For example, if a system limits password length based on the number of bytes in UTF-8, then using non-ASCII characters would consume more bytes and thus the password would be mistakenly truncated. This means that even if the non-ASCII password had more bits of entropy than an ASCII password in the first place, it would now be reduced due to this system bug, thus resulting in a password much easier to guess/crack.

Finally, we noted that in targeted password guessing attacks an attacker can take advantage of the user's nationality, among other characteristics, in order to guess non-English passwords (e.g. words in Greeklish). An attacker can act accordingly and try to guess passwords containing non-English (non-ASCII) characters as well. For example, if an adversary is trying to crack the passwords included in the database of a Greek bank, she may replace some of the English characters with the corresponding ones from the Greek alphabet, like a=α, b=β, w=ω, etc. or use Greek words as a dictionary. More details about how password cracking tools can be used in order to carry out attacks against passwords with non-ASCII characters, are mentioned in the following chapter.

In conclusion, although more research is mandatory on this field, the data we have so far show that the use of non-ASCII characters does not necessarily lead to safer or easier to use passwords. System administrators need to properly inform users about such limitations. Otherwise, users should try and check for themselves whether the system handles passwords as intended, in all cases.

8. Chapter 8: Password Cracking Tools and non-ASCII character passwords

Password-cracking tools like "John the Ripper" and "Hashcat", now allow users to crack passwords that contain characters outside the classic ASCII charset.

In the following paragraphs, trying to stress even more the fact that cracking non-ASCII passwords is not some extremely strenuous process, we describe how we carried out two password cracking attacks using Hashcat against passwords that contained characters from the Greek alphabet.

8.1. Dictionary attack against non-ASCII passwords

During a dictionary attack using Hashcat, all the user needs to do is to make sure the dictionary is in the same encoding as the password was, during the computation of its hash. That is because what Password Cracking Tools actually see, is not the encoding itself but sequences of bytes resulting from the encoding. As we previously noted, the same word can be represented by a different sequence of bytes using different encodings, hence the attack would not be successful if the encoding of the dictionary and the hash did not match.

Since finding out the encoding of the password when the hash was computed is not always feasible, one must take into account the circumstances in which the hash was made. If this is was a hash of a website password for instance, then it is likely that the password is in the same encoding as the website page.

We generated the hash to be cracked on a Linux terminal whose encoding was set to UTF-8 and thus the hash was in UTF-8 encoding as well. Specifically, we produced the MD5 hash of the Greek word 'συνθηματικό' which stands for 'password' in English. In order to get the hashed value of our password we ran the command *"echo -n 'συνθηματικό' | md5sum"* which returned the value: *'eb46c931e0e3db7bb0967e1161152272'*.

As the attack dictionary, we used the "Hunspell" Greek dictionary that contains over 500.000 Greek words which we saved under the name "dictionary.txt", using UTF-8 encoding as well. In order to perform the dictionary attack we composed the following command on our Linux terminal:
"hashcat -m 0 -a 0 eb46c931e0e3db7bb0967e1161152272 ./dictionary.txt".

Unsurprisingly, the password was cracked in just a few seconds, as depicted in Figure 2.

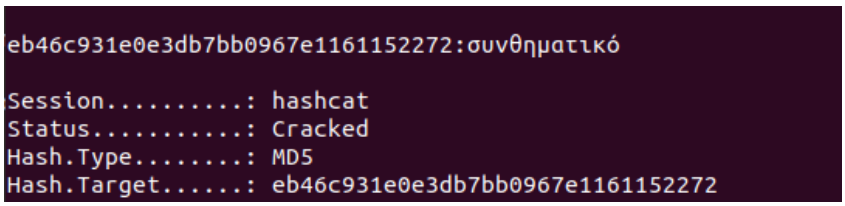

Figure 2: Dictionary attack using Hashcat results

In real life scenarios of course, the adversary can easily replace this Greek dictionary with a large leaked password dataset including passwords in the language of their choice.

8.2. Mask attack against non-ASCII passwords

For the mask attack, we produced the MD5 hash of the Greek word 'ναι' which stands for 'yes' in English, on a Linux terminal using UTF-8 encoding as previously explained during the dictionary attack.

In order to perform the mask attack we composed the following command, which we will refer to as Command-1, on our Linux terminal: *"hashcat -m 0 -a 3 -1 αιν f4e83bae1076624186a6236b3623fc2c ?1?1?1"*. We can break down this command to understand it better. More specifically, '-m 0' means that the hash is MD5, '-a 3' means mask attack, '-1 αιν' is a custom character set including the characters of our password and '?1?1?1' is our mask, which means that we are looking for a string containing three characters from our custom charset. We have deliberately chosen a three letter password as well as a three letter charset, in order to prove our point that is mentioned in the following paragraphs.

When we executed Command-1, our password was not successfully cracked even though it was an easy one and our custom charset contained all of our password's characters, as depicted in Figure 3.

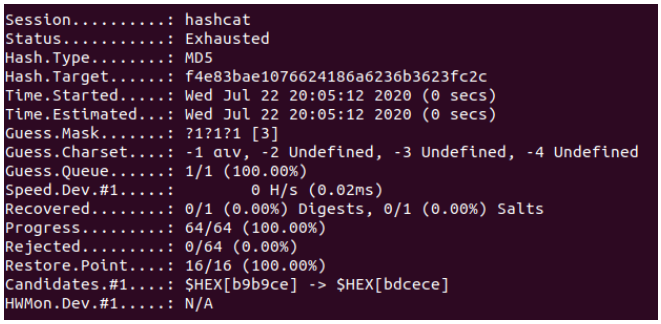

Figure 3: Mask attack using Hashcat – Results of Command-1

If we take a closer look to the results, we can see that sixty-four different passwords were checked in total during the mask attack. Taking into account that we defined a charset and a mask both of three characters in length, then there can only be $3^3 = 27$ different password candidates. This assumption, however, is not entirely accurate because as we have stressed several times so far, what Password Cracking Programs and computers in general actually see, are sequences of bytes and not characters. In UTF-8 encoding, each non-Latin character is represented by not one but two bytes, and therefore when we specified the custom charset '*αιν*', Hashcat saw six bytes in total.

Based on this data we can re-attempt to crack the password, but now while using a six characters mask. Thus, Command-2 is formed as follows: *"hashcat -m 0 -a 3 -1 αιν f4e83bae1076624186a6236b3623fc2c ?1?1?1?1?1?1"*. As we can see in Figure 4, the password is now successfully cracked.

Figure 4: Mask attack using Hashcat – Results of Command-2

Furthermore, if we search for the bytes '*αιν*' is represented by in UTF-8 to get a better understanding of our results, we will get the following sequence: 'ce b1 ce b9 ce bd'. The byte 'ce' appears three time in this sequence, which means we are left with only four unique bytes in a password length of three characters. Thus all the possible combinations are $4^3 = 64$, which explains the number we got when we performed our first mask attack.

As previously mentioned, the reason we defined a custom charset of only three characters, which were the characters of our password in random order, was just to prove our point. In real life scenarios, the adversary will most probably need to use a much larger custom charset. Therefore, after emptying our .pot file we carried-out the mask attack one last time, but now using a custom charset which contained all the letters from the Greek alphabet, i.e. twenty-four characters. As depicted in Figure 5, our password was successfully cracked again.

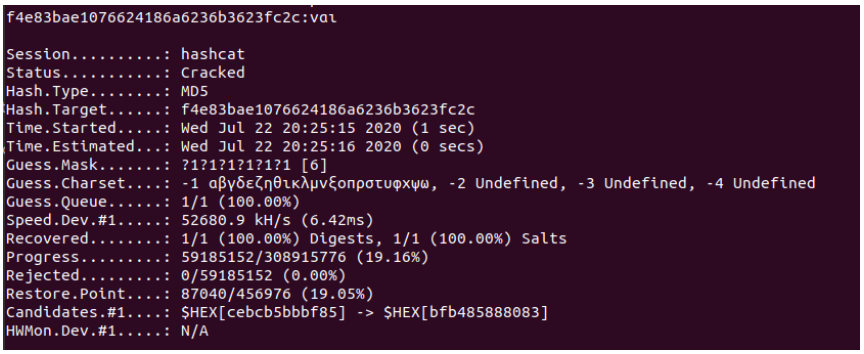

Figure 5: Mask attack using Hashcat – Large custom charset

Conclusion

In the current dissertation, we talked about password strength policies and extensively studied the way some state-of-the-art password strength meters work. We concluded that not all password composition policies achieve their goal of stronger passwords, as they can sometimes result in weaker password distributions that are more susceptible to cracking. The same goes for password strength meters as well. Passwords actually resistant to cracking algorithms, were only created in the presence of stringent meters that also had a variety of visual representations, but mostly when these passwords were used to protect important, high-risk accounts.

Furthermore, we studied about targeted password guessing and concluded that, among others, such attacks can easily be based on the nationality of the users. With this fact in mind, we created a Greeklish password list and carried out password strength scoring sessions using zxcvbn and pcfg_cracker, a pcfg-based password strength meter. We proved that password strength meters tend to inflate the strength score off non-English passwords, mainly because they are trained on leaked password sets of English-speaking users. We then trained both password strength meters using a Greeklish training set, and concluded that for zxcvbn the average score of password strength dropped by about 2 points and that pcfg_cracker was able to guess a significantly larger number of Greeklish passwords in comparison with its default training set.

In addition, we expanded the scope of our research by examining passwords containing non-ASCII characters. We concluded that they are not particularly practical to use, since they require specific input devices and OS settings which may not be available everywhere. Moreover, many modern systems are not able to efficiently support non-ASCII characters due to poor implementation, which causes passwords to be stored as a hash of very few ASCII printable characters or be truncated to only a fraction of the originally submitted non-ASCII characters. Besides implementation bugs which often make non-ASCII passwords less-secure, it has been proved that longer passwords are actually more difficult to crack compared to passwords containing non-English characters or complex combinations.

Finally, we carried out a dictionary and a mask attack against passwords comprised of letters from the Greek alphabet, using Hashcat and actually managed to crack these Greek passwords without much effort. Thus, we

believe that it would most probably be an easy task as well for an adversary trying to crack Greek users' passwords.

References

[1] Maximilian Golla and Markus Dürmuth. 2018. On the Accuracy of Password Strength Meters. In 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS'18), October 15–19, 2018, Toronto, ON, Canada.ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3243734.3243769

[2] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio López. 2012. Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms. In Proceedings of the 2012 IEEE Symposium on Security and Privacy. 523–537.

[3] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. 2011. Of Passwords and People: Measuring the Effect of Password-Composition Policies. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI). ACM, Vancouver, BC, Canada, 2595–2604.

[4] Claude Castelluccia, Markus Dürmuth, and Daniele Perito. 2012. Adaptive Password-Strength Meters from Markov Models. In Symposium on Network and Distributed System Security (NDSS'12). The Internet Society, San Diego, California, USA.

[5] Yimin Guo and Zhenfeng Zhang. 2018. LPSE: Lightweight Password-Strength Estimation for Password Meters. Computers & Security 73 (March 2018), 507–518.

[6] Ding Wang, Debiao He, Haibo Cheng, and Ping Wang. 2016. fuzzyPSM: A New Password Strength Meter Using Fuzzy Probabilistic Context-Free Grammars. In Conference on Dependable Systems and Networks (DSN '16). IEEE Computer Society, Toulouse, France, 595–606.

[7] Daniel Lowe Wheeler. 2016. zxcvbn: Low-Budget Password Strength Estimation. In USENIX Security Symposium (SSYM'16). USENIX, Austin, Texas, USA, 157–173.

[8] Xavier de Carné de Carnavalet and Mohammad Mannan. 2014. From Very Weak to Very Strong: Analyzing Password-Strength Meters. In Symposium on Network and Distributed System Security 2014 (NDSS'14). The Internet Society, San Diego, California, USA

[9] William Melicher, Blasé Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2016. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In USENIX Security Symposium (SSYM '16). USENIX, Austin, Texas, USA, 175–191.

[10] William E. Burr, Donna F. Dodson, and W. Timothy Polk. 2004. Electronic Authentication Guideline: NIST SP 800-63 Ver. 1.0 (2004) to 800-63-2 (2013). https://csrc.nist.gov/publications/detail/sp/800-63/ver-10/archive/2004-06-30, as of September 10, 2018.

[11] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan and Xinyi Huang. 2016. Targeted Online Password Guessing: An Underestimated Threat. In CCS '16, Vienna, Austria, DOI: http://dx.doi.org/10.1145/2976749.2978339

[12] Houshmand S., Aggarwal S. (2017) Using Personal Information in Targeted Grammar-Based Probabilistic Password Attacks. In: Peterson G., Shenoi S. (eds) Advances in Digital Forensics XIII. Digital Forensics 2017. IFIP Advances in Information and Communication Technology, vol 511. Springer, Cham

[13] Blasé Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, Noah Johnson, and William Melicher. 2017. Design and Evaluation of a Data-Driven Password Meter. In ACM Conference on Human Factors in Computing Systems (CHI '17). ACM, Denver, Colorado, USA, 3775–3786.

[14] Y. Li, H. Wang, and K. Sun, Study of Personal Information in Human-chosen Passwords and Its Security Implications, IEEE INFOCOM, 2016.

[15] M. Dürmuth, A. Chaabane, D. Perito, and C. Castelluccia. 2013. When privacy meets security: Leveraging personal information for password cracking, CoRR abs/1304.6584, 2013

[16] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation. In USENIX Security Symposium (SSYM '12). USENIX, Bellevue, Washington, USA, 65–80.

[17] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. 2013. Does My Password Go Up to Eleven?

The Impact of Password Meters on Password Selection. In ACM Conference on Human Factors in Computing Systems (CHI '13). ACM, Paris, France, 2379–2388.

[18] Jeremy Hsu, How Language Shapes Password Security. IEEE Spectrum. 25 September 2019. https://spectrum.ieee.org/tech-talk/telecom/security/how-language-shapes-chinese-and-english-password-security. (Online; accessed 14/12/2019).

[19] Mashael AlSabah, Gabriele Oligeri, Ryan Riley. Your Culture is in Your Password: An Analysis of a Demographically-diverse Password Dataset. Computers & Security Vol. 77, 427-441, Elsevier, August 2018.

[20] Zhigong Li, Weili Han. 2014. A Large-Scale Empirical Analysis of Chinese Web Passwords. In USENIX Security Symposium (SSYM '14). USENIX, San Diego, California, USA.

[21] Ding Wang, Ping Wang, Debiao He and Yuan Tian. 2019. Birthday, Name and Bifacial-security: Understanding Passwords of Chinese Web Users. In USENIX Security Symposium (SSYM '19). USENIX, Santa Clara, California, USA.

[22] Artemios G. Voyiatzis, Christos A. Fidas, Dimitrios N. Serpanos and Nikolaos M. Avouris. 2011. An Empirical Study on the Web Password Strength in Greece. In the 15th Panhellenic Conference on Informatics. Kastoria, Greece. Added to IEEE Xplore on 03 November 2011.

[23] George E. Violettas and Kyriakos Papadopoulos. 2014. Passwords to absolutely avoid. In the Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014). Bangalore, India. Added to IEEE Xplore on 15 May 2014

[24] Iraklis Mathiopoulos. Creating a Greeklish wordlist for password cracking. Medium. 23 July 2016. https://medium.com/@iraklis/creating-a-Greeklish-wordlist-for-password-cracking-2ebd756a87a9 . (Online; accessed 14/12/2019).

[25] Matt Weir. Probabilistic Context Free Grammar (PCFG) password guess generator. https://github.com/lakiw/pcfg_cracker. (Online; accessed 01/04/2020).

[26] Kranias Orestis. Greeklish Password Wordlists. https://github.com/kraniasorestis/Greek-wordlists. (Online; accessed 10/05/2020).

[27] Bonneau J. and Xu R. 2012. Of contraseñas, סיסמאות, and 密密密码码码. Character encoding issues for web passwords. Web 2.0 Security & Privacy

[28] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2014. Can Long Passwords Be Secure and Usable?. CHI 2014, Apr 26 - May 01 2014, Toronto, ON, Canada

[29] Matt Weir, Sudhir Aggarwal, Breno de Medeiros and Bill Glodek. 2009. Password Cracking Using Probabilistic Context-Free Grammars. 30th IEEE Symposium on Security and Privacy. Berkeley, CA, USA. Added to IEEE Xplore on 18 August 2009

[30] Arvind Narayanan and Vitaly Shmatikov. 2005. Fast dictionary attacks on passwords using time-space tradeoff. In Proceedings of the 12th ACM conference on Computer and communications security (CCS '05). Association for Computing Machinery, New York, NY, USA, 364–372