

When Privacy meets Security: Leveraging personal information for password cracking

M. Dürmuth¹, A. Chaabane², D. Perito², and C. Castelluccia²

¹ Ruhr-University Bochum

`markus.duermuth@rub.de`

² INRIA, France

`firstname.lastname@inria.fr`

Abstract. Passwords are widely used for user authentication and, despite their weaknesses, will likely remain in use in the foreseeable future. Human-generated passwords typically have a rich structure, which makes them susceptible to guessing attacks. In this paper, we study the effectiveness of guessing attacks based on Markov models. Our contributions are two-fold. First, we propose a novel password cracker based on Markov models, which builds upon and extends ideas used by Narayanan and Shmatikov (CCS 2005). In extensive experiments we show that it can crack up to 69% of passwords at 10 billion guesses, more than all probabilistic password crackers we compared against. Second, we systematically analyze the idea that additional personal information about a user helps in speeding up password guessing. We find that, on average and by carefully choosing parameters, we can guess up to 5% more passwords, especially when the number of attempts is low. Furthermore, we show that the gain can go up to 30% for passwords that are actually based on personal attributes. These passwords are clearly weaker and should be avoided. Our cracker could be used by an organization to detect and reject them. To the best of our knowledge, we are the first to systematically study the relationship between chosen passwords and users' personal information. We test and validate our results over a wide collection of leaked password databases.

1 Introduction

Password-based authentication is the most widely used form of user authentication, both online and offline. Passwords will likely remain the predominant form of authentication for the foreseeable future, due to a number of advantages: passwords are highly portable, easy to understand for laypersons, and easy to implement for the operators. Despite the weaknesses passwords have, they still are and will be in use for some time. The reason can be found in [1], which lists a large number of criteria that user authentication may fulfill, and measures the quality of a large number of user authentication mechanisms. Alternative forms of authentication can complement password, but have not been able, so far, to provide a standalone alternative solution.

In this work, we concentrate on offline guessing attacks, in which the attacker can make a number of guesses bounded only by the time and resources she is willing to invest. While such attacks can be improved by increasing the speed with which an attacker can make guesses (e.g., by using specialized hardware and large computing resources [5, 4]), we concentrate here on techniques to reduce the number of guesses required to crack a password. Hence, our approach reduces the attack time independently of the available resources.

The optimal strategy for password cracking (both offline and online) is to enumerate passwords in decreasing order of likelihood, i.e., trying more frequent passwords first and less frequent passwords later. Moreover, human chosen passwords frequently have a rich structure which can be exploited to generate candidate guesses (e.g., using a dictionary and a set of concatenation rules).

Tools commonly used for password cracking, such as John the Ripper (JtR), exploit regularities in the structure of password by applying *mangling rules* to an existing dictionary of words (e.g., by replacing the letter **a** with **@** or by appending a number). Such approach, allows to generate new guesses from an existing corpus of data, like a dictionary or a previously leaked password database. Recent work [11][13] has shown ways to improve cracking performance by enumerating guessed passwords based on their likelihood. These password crackers have been shown to outperform JtR in certain conditions. The first insight of our work will be to build upon and improve on the performance of these probabilistic password crackers. Furthermore, while previously proposed password crackers outperform JtR, they do not consider any *user specific* information. This means that the guesses outputted by each of these tools are fixed and do not depend upon additional information about the user.³

Common sense would suggest that guessing a password can be done more efficiently when personal information about the victim is known. For example, one could try to guess passwords that contain the victim’s date of birth or the names of their siblings. However, this raises the question, how do we order (in a probabilistic sense) what personal information to include in the guessing? While guessing likely passwords, shall we include all the information about the victims or only restrict the attack to a subset of that information? To the best of our knowledge, these questions have not been thoroughly explored so far and, as we will show, have a serious impact on the security of passwords. This is especially true since the steadily increasing use of social networks gives attackers access to a vast amount of public information about their victims for the purpose of password cracking.

Paper organization: We review some basics on password guessing, commonly used password guessers, as well as more related work in Section 2. In Section 3 we describe the *Ordered Markov ENumerator* (OMEN), and compare its performance with other password guessers. Section 4 details the idea of exploiting personal information in password guessing, some basic statistics about the data we use, a detailed description of our algorithm and the results of our experiments. We finally conclude this paper with a discussion of our findings.

³ JtR does some very limited guessing depending on the username

2 Related Work

One of the main problems with passwords is that many users choose *weak* passwords. These passwords typically have a rich structure and thus can be guessed much faster than with brute-force guessing attacks. Best practice mandates that only the hash of a password is stored on the server, not the password, in order to prevent leaking plain-text when the database is compromised. Furthermore, additional *salting* is used to avoid pre-computation attacks. Let H be a hash function and pwd the password, choose a random bitstring $s \in_{\mathcal{R}} \{0, 1\}^{16}$ as salt and store the tuple $(s, h = H(pwd \parallel s))$. In this work we mainly consider *offline guessing attacks*, where an attacker is given access to the tuple (s, h) , and tries to recover the password pwd . The hash function is frequently designed for the purpose of slowing down guessing attempts [9]. This means that the cracking effort is *strongly dominated by the computation of the hash function* making the cost of generating a new guess relatively small. Therefore, we evaluate all password crackers based on the number of attempts they make to correctly guess passwords.

2.1 John the Ripper

John the Ripper (JtR) [7] is one of the most popular password crackers. It proposes different methods to generate passwords: In *dictionary* mode, a dictionary of words is provided as input, and the tool tests each one of them. Users can also specify various mangling rules. When mangling rules are provided, JtR applies each rule to each word in the input dictionary. In real attacks, the dictionary mode using simple mangling rules works surprisingly well, especially when the input dictionary is derived from large collections of leaked passwords. Similarly to [3], we discover that for relatively small number of guesses (less than 10^8), JtR in dictionary mode produces best results. However, we focus on attacks with larger number of attempts, for which simple, non probabilistic approaches fall short.

2.2 Password Guessing with Markov Models

Markov models have proven very useful for computer security in general and for password security in particular. They are an effective tool to crack passwords [6], and can likewise be used to accurately estimate the strength of new passwords [2].

The underlying idea is that adjacent letters in human-generated passwords are not independently chosen, but follow certain regularities (e.g., the 2-gram **th** is much more likely than **tq** and the letter **e** is very likely to follow **th**). In an n -gram Markov model, one models the probability of the next character in a string based on a prefix of length $n - 1$. Hence, for a given string c_1, \dots, c_m , a Markov model estimates its probability as

$$P(c_1, \dots, c_m) = P(c_1, \dots, c_{n-1}) \cdot \prod_{i=n}^m P(c_i | c_{i-n+1}, \dots, c_{i-1}). \quad (1)$$

For password cracking, one basically learns the initial probabilities $P(c_1, \dots, c_{n-1})$ and the transition probabilities $P(c_n | c_1, \dots, c_{n-1})$ from real-world data (which should be as close as possible to the distribution we expect in the data that we attack), and then enumerates passwords in order of descending probabilities as estimated by the Markov model (according to Equation 1).

To make this attack efficient, we need to consider a number of details. First, one usually has a limited dataset when learning the initial probabilities and transition probabilities. The limited data entails that one cannot learn frequencies with arbitrarily high accuracy, i.e., *data sparseness* is a problem. The critical parameters are the size of the alphabet Σ and the parameter n , which determines the length of the n -grams.

Second, one needs an algorithm that *enumerates the passwords* in the right order. While it is easy to compute the probability for a given password, it is not clear how to enumerate the passwords in decreasing probability. To overcome this problem, [6] provides a method to enumerate all passwords that have probability larger than a given threshold λ , but not necessarily in descending order. Hence, *all* passwords that have a probability (approximately) higher than λ are produced in output, where λ is an input parameter. This is sufficient for attacks based on precomputation (rainbow tables), but not for “normal” guessing attacks where guessing passwords in the correct order can drastically reduce guessing time.

Notably, an add-on to JtR has been released with an independent implementation of the algorithm presented in [6]. The implementation is available at [7]. In the evaluation section, we use this implementation as a comparison (and refer to as JtR-markov).

JtR Incremental mode (JtR-inc) [7]: The *incremental* mode tries passwords based on a (modified) 3-gram Markov model. Specifically, JtR-inc computes not only the probability of each 3-gram but also the probability that this particular 3-grams appears at certain indices. In this way, this attack takes into account the structure of the password (e.g., upper case appears usually at the front while numbers at the end). However, two key differences are to note: first, as for Narayanan and al. algorithm, the guesses are not generated in the *true* probability order and second, the Markov chain is modified so that it can cover the entire keyspace.

2.3 Probabilistic Grammars-based Schemes

Probabilistic context-free grammar (PCFG) schemes make the assumption that password structures have different probabilities [11]. In other words, some structures, such as passwords composed of 6 letters followed by 2 digits, are more frequent than others. The main idea of these schemes is therefore to extract the most frequent structures, and use them to generate password candidates.

More precisely, in the *training phase*, different structures are extracted from lists of real-world passwords, where each structure indicates the positions of lower and uppercase letters, numerical, and special characters, as well as the associated probabilities. In the *attack phase (or password generation phase)*, an algorithm

outputs the possible structures for the grammar with decreasing probabilities. From this output, which describes positions of the four classes of characters, password guesses are generated as follows: Numerical and special characters are substituted by those that have been observed in the training phase and in decreasing order of probability, and letters are substituted with appropriate words from a dictionary. This gives the final password candidate list.

3 OMEN: An Improved Markov Model Based Password Cracker

In this section we present a more efficient implementation of password enumeration based on Markov models, which is the first contribution of our work. Our implementation improves previous work based on Markov models by Narayanan et al. [6] and JtR [7]. Note that the indexing algorithm presented by Narayanan et al. [6] combines two ideas: first, it uses Markov models to index only passwords that have high probability, and second, it utilizes a hand-crafted finite automata to accept only passwords of a specific form (e.g., eight letters followed by a digit). Our algorithm is solely based on Markov models to create guesses. However, it could be combined with similar ideas as well.

3.1 An Improved Enumeration Algorithm

Narayanan et al.’s indexing algorithm [6] has the disadvantage of not outputting passwords in order of decreasing probability, however, guessing passwords in the right order can substantially speed up password guessing (see the example in Section 3.2). We developed an algorithm, the *Ordered Markov ENumerator* (OMEN), to enumerate passwords with (approximately) decreasing probabilities.

On a high level, our 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. More precisely, we first take the logarithm of all n -gram probabilities, and discretize them into levels (denoted η) similarly to Narayanan et al. [6], according to the formula $lv_i = \text{round}(\log(c_1 \cdot prob_i + c_2))$, where c_1 and c_2 are chosen such that the most frequent n -grams get a level of 0 and that n -grams that did not appear in the training are still assigned a small probability. Note that levels are negative, and we adjusted the parameters to get 10 different levels, i.e., the levels can take values $0, -1, \dots, -9$. The number of levels influences both the accuracy of the algorithm as well as the runtime: more levels means better accuracy, but increased runtime.

For a specific length ℓ and level η , $\text{enumPw}(\eta, \ell)$ proceeds as follows:

1. It identifies all vectors $\mathbf{a} = (a_2, \dots, a_\ell)$ of length $\ell - 1$ (when using 3-grams we need $\ell - 2$ transition probabilities and 1 initial probability to determine the probability for a string of length ℓ), such that each entry a_i is an integer in the range $[0, -9]$, and the sum of all elements is η .

Algorithm 1 Enumerating passwords for level η and length ℓ (here for $\ell = 4$).⁴

function enumPwd(η, ℓ)

1. for each vector $(a_i)_{2 \leq i \leq \ell}$ with $\sum_i a_i = \eta$
 and for each $x_1x_2 \in \Sigma^2$ with $L(x_1x_2) = a_2$
 and for each $x_3 \in \Sigma$ with $L(x_3 \mid x_1x_2) = a_3$
 and for each $x_4 \in \Sigma$ with $L(x_4 \mid x_2x_3) = a_4$:
 (a) output $x_1x_2x_3x_4$
-

2. For each such vector \mathbf{a} , it selects all 2-grams x_1x_2 whose probabilities match level a_2 . For each of these 2-grams, it iterates over all x_3 such that the 3-gram $x_1x_2x_3$ has level a_3 . Next, for each of these 3-grams, it iterates over all x_4 such that the 3-gram $x_2x_3x_4$ has level a_4 , and so on, until the desired length is reached. In the end, this process outputs a set of candidate passwords of length ℓ and level (or “strength”) η .

A more formal description is presented in Algorithm 1. It describes the algorithm for $\ell = 4$. However, the extension to larger ℓ is straightforward.

Example: We illustrate the algorithm with a brief example. For simplicity, we consider passwords of length $\ell = 3$ over a small alphabet $\Sigma = \{a, b\}$, where the initial probabilities have levels

$$\begin{aligned} L(aa) &= 0, & L(ab) &= -1, \\ L(ba) &= -1, & L(bb) &= 0, \end{aligned}$$

and transitions have levels

$$\begin{aligned} L(a|aa) &= -1 & L(b|aa) &= -1 \\ L(a|ab) &= 0 & L(b|ab) &= -2 \\ L(a|ba) &= -1 & L(b|ba) &= -1 \\ L(a|bb) &= 0 & L(b|bb) &= -2. \end{aligned}$$

- Starting with level $\eta = 0$ gives the vector $(0, 0)$, which matches to the password **bba** only (the prefix “aa” matches the level 0, but there is no matching transition with level 0).
- Level $\eta = -1$ gives the vector $(-1, 0)$, which yields **aba** (the prefix “ba” has no matching transition for level 0), as well as the vector $(0, -1)$, which yields **aaa** and **aab**.
- Level $\eta = -2$ gives three vectors: $(-2, 0)$ yields no output (because no initial probability matches the level -2), $(-1, -1)$ yields **baa** and **bab**, and $(0, -2)$ yields **bba**.
- and so one for all remaining levels.

The selection of ℓ (i.e. the length of the password to be guessed) is challenging, as the frequency with which a password length appears in the training data

⁴ Here $L(xy)$ and $L(z|xy)$ stand for the level of initial and transition probabilities, respectively.

is not a good indicator of how often a specific length should be guessed. For example, assume that there are as many passwords of length 7 and of length 8, then the success probability of passwords of length 7 is larger as the search-space is smaller. Hence, passwords of length 7 should be guessed first. Therefore, we use an adaptive algorithm that keeps track of the success ratio of each length and schedules more passwords to guess for those lengths that were more effective. More precisely, our adaptive password scheduling algorithm works as follows:

1. For all n length values of ℓ (we consider lengths from 3 to 20, i.e. $n = 17$), execute $\text{enumPwd}(0, \ell)$ and compute the success probability $sp_{\ell,0}$. This probability is computed as the ratio of successfully guessed passwords over the number of generated password guesses of length ℓ .
2. Build a list L of size n , ordered by the success probabilities, where each element is a triple $(sp, level, length)$. (The first element $L[0]$ denotes the element with the largest success probability.)
3. Select the length with the highest success probability, i.e., the first element $L[0] = (sp_0, level_0, length_0)$ and remove it from the list.
4. Run $\text{enumPwd}(level_0 - 1, length_0)$, compute the new success probability sp^* , and add the new element $(sp^*, level_0 - 1, length_0)$ to L .
5. Sort L and go to Step 3 until L is empty or enough guesses have been made.

3.2 Performance Evaluation

In this section, we present a comparison between our improved Markov model password cracker and previous state-of-the-art solutions.

Datasets: We evaluate the performance of our password guesser on multiple datasets. One of the largest lists currently publicly available is the *RockYou list* (RY), consisting of 32.6 million passwords that were obtained by an SQL injection attack in 2009. The passwords were leaked in clear, all further information was stripped from the list before it was leaked to the public. This list has two advantages: first, its large size gives well-trained Markov models; second, it was collected via an SQL injection attack therefore affecting all the users of the compromised service. We split the RockYou list into two subsets: a *training set* (RY-t) of 30 million and a *testing set* (RY-e) of the remaining 2.6 million passwords.

The *MySpace list* (MS) contains about 50 000 passwords (different versions with different sizes exist, most likely caused by different sanitation or leaked from the servers at different points in time). The passwords were obtained in 2006 by a phishing attack. As before, we split the list in a *training set* (MS-t) of 30 000 and a *testing set* (MS-e) of the remaining 20 000 passwords.

The *Facebook list* (FB) was posted on the pastebin.com website⁵ in 2011. This dataset contains both Facebook passwords and associated email addresses. It is unknown how the data was obtained by the hacker, but most probably was collected via a phishing attack. Finally, we used a list of 60 000 email addresses

⁵ <http://pastebin.com/>

and passwords leaked by the group LulzSec (we call this list LZ). The list was publicly released in June 2011 via Twitter⁶.

We complemented the Facebook list by collecting the public information associated to the Facebook profiles connected to the email addresses. For each profile, we collected the public attributes, which include: first/last name; location; date of birth; friends names; siblings names; education/work names.

Ethical Considerations: Leaked password databases have been used in a number of studies on passwords [11, 10, 2]. Studying databases of leaked password has arguably helped the understanding of users real world password practices. The information we used in our study was already available to the public.

Results: In this section, we evaluate the efficiency of our password guesser OMEN, and compare it with other password guessers on different datasets. We discover that OMEN has consistently better performance compared to previously proposed algorithms. For the experiments, we trained OMEN using the RockYou training set RY-t (or MS-t in one experiment), and evaluated it on the sets RY-e, MS (or MS-e when training on MS-t), and FB. Table 1 provides a summary of the results. The table can also be used as a comparison of all the previously proposed password crackers.

Algorithm	Training Set	Testing Set		
		RY-e	MS-e	FB
Omen	RY-t (10^{10})	69%	66%	64%
	RY-t	60%	54%	54%
	MS-t ($0.8 * 10^{10}$)	64%	68%	49%
PCFG [11]	MS-t ($0.8 * 10^{10}$)	37%	53%	29%
JtR-Markov [6]	RY-t (10^{10})	64%	60%	61%
	RY-t	53%	40%	50%
JtR-Inc	RY-t	44%	37%	40%

Table 1: Summary table indicating the percentage of cracked passwords for 1 billion guesses (or 10 billion when specified).

OMEN vs JtR’s Markov Mode: Figure 1a shows the comparison of OMEN and the Markov mode of JtR. JtR’s Markov mode implements the password indexing function by Narayanan et al. [6]. Both models are trained on a list of passwords (*-t). Then, given a target number of guesses T (here 1 billion), we computed the corresponding level (η) to output T passwords. The curve shows the dramatic improvement in cracking *speed* given by our improved ordering of the password guesses. In fact, JtR-Markov outputs guesses in no particular order which implies that likely passwords can appear “randomly” late in the guesses. This behaviour leads to the near-linear curves shown in Figure 1a. One may ask whether JtR-Markov would surpass OMEN after the point T ; the answer is *no* as the results do not extend linearly beyond the point T ; and larger values of T lead to a flatter curve. To demonstrate this claim, we performed the same

⁶ <https://twitter.com/#!/LulzSec/status/81327464156119040>

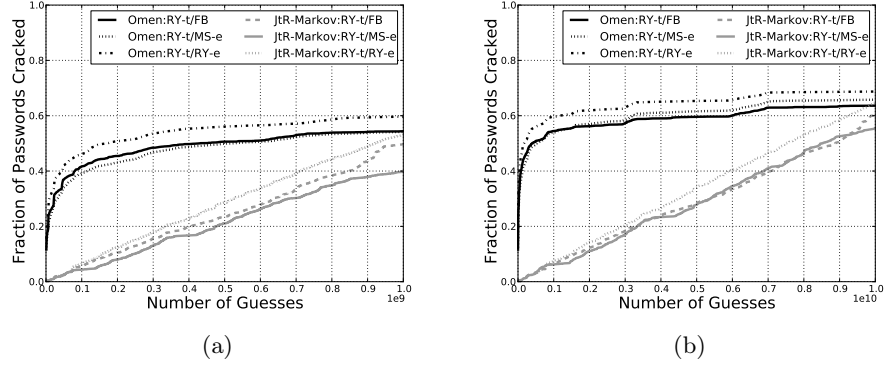


Fig. 1: (a) Comparing OMEN with the JtR Markov mode, 1B guesses. (b) OMEN Vs JtR Markov mode, 10 billion guesses

experiment with T equals to 10 billion guesses (instead of 1 billion). Figure 1b shows how the linear curve becomes *flatter*.

To show the generality of our approach, we compare the cracking performance on three different datasets: RY, FB and MS. The ordering advantage allows OMEN to crack more than 40% of passwords (independently of the dataset) in the first 90m guesses while JtR-Markov cracker needs at least eight times as many guesses to reach the same goal.

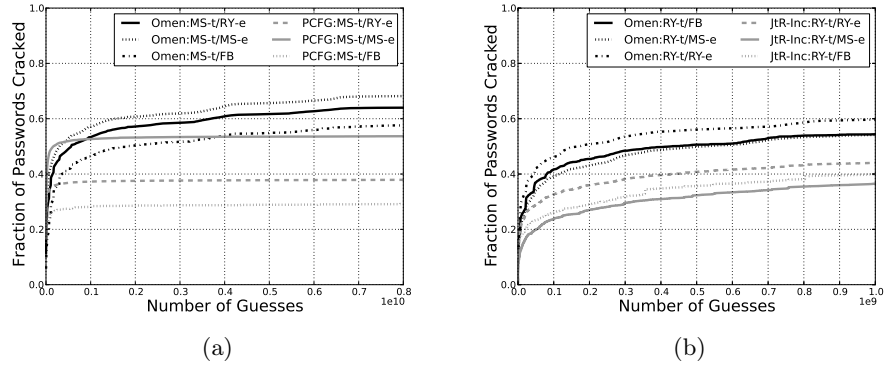


Fig. 2: Comparing OMEN to the PCFG guesser (a) and to JtR incremental mode(b).

OMEN vs PCFG: Figure 2a compares our guesser with the PCFG password guesser of Weir et al. [11], based on the code available in [8]. We run it using the configuration as described in the paper and using the dictionary dict-0294 [12]. In this experiment we trained on the MySpace dataset, since the public version of PCFG seemed unable to correctly train on the larger RockYou dataset due to a memory bug.

OMEN outperforms the PCFG guesser, except when testing on the MySpace list, where PCFG produces slightly better guesses for the first 100 million attempts. However, OMEN produces better guesses after 100 million and outperforms PCFG by around 10% at 8 billion guesses. We believe the reason is that the grammar for PCFG is trained on a subset of the MySpace list, which is better adapted to guessing the MySpace list, whereas the MySpace list is too small to meaningfully train Markov models. Note that PCFG mostly plateaus after 0.5 billion guesses and results hardly improve any more, whereas OMEN still produces noticeable progress. For the other testing sets, however, OMEN produces better guesses almost from the beginning.

OMEN vs JtR’s Incremental Mode: We also compare OMEN to JtR in incremental mode (Figure 2b). Similarly to the previous experiments, both crackers were trained on the RockYou training set of 30 million passwords. It appears clear that the incremental mode in JtR produces worse guesses than OMEN. Notably, it also produces worse guesses than any other cracker tested.

4 Personal Information and Password Guessing

The results of the previous section show that a significant fraction of passwords can be guessed with a (relatively) moderate number of attempts, compared to the entire possible search space. However, most techniques adopt a *coarse grained* approach that relies on a *generic* probability distribution, which by definition, does not depend on the password being guessed. Intuitively, exploiting personal information in the password cracking process may enhance the success ratio. Surprisingly, such possibility has not been extensively studied.

While a multitude of personal information can be used, we focus on a realistic scenario where these information can easily be extracted from a *public* source. For instance, an attacker armed with his victim email address, can gather her social network profile and use the collected information to guess her password. Such information includes:

- Information related to the *user’s name*, such as first name, last name, user-name;
- *Social relations* such as friends’ and family members’ names;
- *Interests* such as hobbies, favorite movies, etc;
- *Location information* like the place of residence;

In the next sections, we explore the relationship between such information, referred to as *hint* in the rest of this paper, and passwords, as well as its effect on password cracking.

4.1 Similarity between Passwords and Personal Information

To assess whether social information can be exploited to improve password cracking, we quantify the correlation between passwords and personal information. We

use 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. For example, the LCSS of the two strings `abcabc` and `abcba` is `abc`, and the LCSS of `abcabc`.

(Modified) Jaccard similarity (JS): The Jaccard similarity index compares similarity of two sets. For two sets X and Y , the Jaccard index is $J(X, Y) := \frac{|X \cap Y|}{|X \cup Y|}$. It is a similarity measure on sets, not on strings. However, we extract the n -grams from a string and apply the JS function to the sets of n -grams. There is one drawback of this measure that does not match our application, namely that appending unrelated information to the hint (which degrades the “real” usefulness only for large appended text) rapidly decreases the JS value. Therefore, we use a “modified JS” defined as follows: Given a password P and a *hint* H , and denoting the set of 3-grams that appear in P and H with P_{3g} and H_{3g} , respectively, we define $J_{3g}^*(P, H) := \frac{|P_{3g} \cap H_{3g}|}{|P_{3g}|}$. Figure 3 displays the cumulative distribution function (CDF) of the JS between passwords and personal information for FB and LZ datasets. For each password of the Facebook dataset, we compute the JS with each of its corresponding personal attribute.

The lowest (green) plot (FB Max) displays the CDF of the maximum of these values. It basically shows that about 35% of Facebook passwords are somewhat correlated with one of their user’s attributes. Note that any non-zero similarity means that at least one 3-gram is shared, which already is a substantial overlap. The correlation becomes stronger for about 10% of users. The grey (FB Username) and black (LZ Username) curves display the CDF of the similarity between passwords and *UserNames* for the FB and LZ datasets, respectively. They both show similar shape, although surprisingly Facebook passwords seem to be more correlated to *UserNames* than LZ passwords. In both datasets, about 10% of passwords seem to be highly correlated with the *UserNames* attribute.

Table 2 goes into more details and summarizes similarity measures between different attributes and the respective passwords of the FB datasets. As shown by Figure 3, more than 80% of passwords have little similarity with personal information attributes. This explains the small similarity values in column JS and LCSS, that correspond to averages of the similarities over the whole dataset (since many similarities are equal to zero, the resulting averages have pretty low values). For this reason, we order the passwords according to their similarity

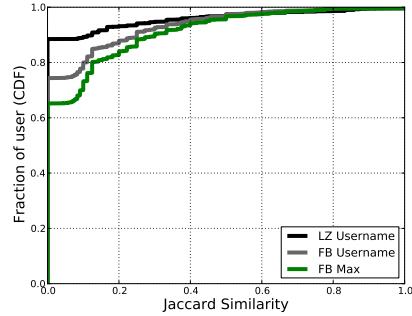


Fig. 3: CDF of Jaccard similarity

Attribute	JS	JS(5%)	LCSS	LCSS(5%)	Len
FirstName	0.02	0.31	0.93	4.34	5.84
LastName	0.01	0.24	0.71	3.55	6
Username	0.07	0.58	1.48	6.31	10.53
Friends	0.06	0.30	1.54	4.15	147.28
Edu/Work	0.02	0.23	1.20	3.5	40.93
Contacts	0.06	0.63	1.44	6.55	17.67
Location	0.01	0.13	1.07	2.94	25.70
Birthday	0.04	0.5	0.87	4	6.84
Siblings	0.04	0.36	1.27	4.96	94.71

Table 2: Mean similarity between passwords and personal information (FB dataset).

values for the different attributes. We then present, in the columns JS(5%) and LCSS(5%), the average similarity of the top 5% for each attributes.

First we notice how attributes such as *UserNames*, *FirstName* and *Birthday* seem to substantially overlap with the passwords. For instance, for the top 5% users, *UserNames* and *Birthday* share half of the n -grams with the password and have more than 4 and 6.4 common substring with it respectively. Furthermore, we notice that long attributes such as *Friends* or *Education* and *Work* (on average 150 characters long and 50, respectively) have a high value of LCSS. Finally, the LCSS(5%) results show that the average LCSS value is around 3 which sustains the usage of a 3-grams model (rather than 2-grams or 4-grams).

In Section 4.2 we will explore how to incorporate these findings to robustly increase the performance of OMEN.

Password Creation Policies and Usernames One surprising fact highlighted by the data in the previous section is that usernames and passwords are very similar in a small, yet significant, fraction of the cases. This fact prompted us to study this specific aspect of password policies in depth, reader may refer to Appendix A for more details. The results are worrisome: out of 48 tested sites, 27 allowed *identical* username and password, including major sites such as Google, Facebook, and Amazon, and only 4 sites required more than one character difference between the two. This could lead to highly effective guessing attack.

4.2 OMEN+: Improving OMEN Performance with Personal Information

In Section 4.1 we showed that users’ personal information and passwords can be correlated. However, it is not clear how to use such information when guessing passwords especially that some information may have a negatively impact on the performance. Let us illustrate this possibility with an example: assume that we possess extensive information about a victim. This information may include name, date of birth, location information, family member names, etc. Intuitively, this information should increase the performance of a password cracker.

For example, we could generate a password guess with the name of a sibling concatenated with their year of birth. However, in order to increase the (overall) performance, one must still order password guesses in decreasing probability. Otherwise, the integration of additional information can decrease effectiveness. To show this, for the sake of argument, let us assume that the attacker is only allowed *one* guess. The same argument can be extended to any number of guesses. The attacker should use some personal information for the one guess only if the probability of this password is higher than the most frequent “generic” password, say 123456. Assuming that no user specific password is, on average, more frequent than 123456, then, by including personal information, the attacker would decrease her chances of success for the single guess.

Boosting Algorithm: It is challenging to decide how to use the additional personal information. In fact, only certain parts of this data overlap with the password. In a dictionary-based attack, we need to decide which substring(s) should be added to the attack dictionary, and choosing the wrong ones one could decrease performance. With Markov models, however, the situation is easier, as n -grams are a canonical target. By increasing (conditional) probabilities of important n -grams (i.e. n -grams that are contained in *hints*), we can increase the probability of passwords that are related to them, and thus improve OMEN’s performance. Our boosting algorithm takes as input a parameter $\alpha > 1$ (see Section 4.2 on how this parameter is chosen), a hint h , and a Markov model consisting of the initial probabilities $p(xy)$ and the conditional probabilities $p(z|xy)$, and outputs modified conditional probabilities $p^*(z|xy)$. Let us assume we have a list of N passwords pwd_1, \dots, pwd_N , and for each password pwd_i we have some additional information $hint_i$, which may or may not help us in guessing the password. We want to automatically and efficiently determine if a specific set of hints is useful or not, and how strongly each of the hints should be weighted. Note that since hints are password-specific, trying all possible combinations would be too computationally expensive. Our algorithm works as follows⁷:

1. For each pair $pwd_i, hint_i$, two sets are defined: S_i is the set of 3-grams that appear in both the password and the hint and T_i is the set of 3-grams from pwd_i such that $hint_i$ has a 3-gram that shares the first two letters, but not the third. For instance, if $pwd_i = \text{password}$ and $hint_i = \text{passabcd}$ then $S_i = \{\text{pas}, \text{ass}\}$ and $T_i = \{\text{ssw}\}$.
2. For each 3-gram xyz in S_i , we set the conditional probability

$$p^*(z|xy) := \alpha \cdot p(z|xy)$$

for a given parameter α . Considering the previous example, we boost the 3-grams **pas** and **ass**, as follows: $p^*(s|pa) := \alpha \cdot p(s|pa)$; $p^*(s|as) := \alpha \cdot p(s|as)$. By modifying a conditional probability from \hat{p} to $\alpha \cdot \hat{p}$ we distribute a probability mass of $(\alpha - 1)\hat{p}$ that we need to subtract at another place. The probability \hat{p} is (in practice) much smaller than 1 (we use an alphabet

⁷ To ease presentation, we only describe the estimation algorithm for 3-grams. The generalization to n -grams is straightforward.

size of $|\Sigma| = 72$), so $1 - \hat{p} \approx 1$. Consequently, if we multiply all remaining (conditional) probabilities except \hat{p} with $(1 - \alpha\hat{p})$, they sum up to approximately 1 again (using the approximation simplifies the calculations): $(1 - \alpha\hat{p}) \cdot (\sum_{i \neq z} p(i|xy)) + \alpha\hat{p} \approx (1 - \hat{p}\alpha) \cdot 1 + \alpha\hat{p} = 1$. Writing $s_i := |S_i|$ and $t_i := |T_i|$ for the sizes of the two sets, the overall effect on password probabilities is

$$p_{pwd_i}^* = \prod_{i \in (pwd_i)_{3g}} p_i \approx \alpha^{s_i} (1 - \hat{p}\alpha)^{t_i} \cdot p_{pwd_i}^{old}$$

where $p_{pwd_i}^*$ is the “new” probability after boosting the n -grams, and $p_{pwd_i}^{old}$ is the “old” probability before boosting.

Estimating Boosting Parameters The section describes how the boosting parameter α of each $hint_i$ is computed. Recall that OMEN outputs password guesses in descending order of their (estimated) probabilities. Let f denote the function that gives the estimated probabilities $x = f(y)$ for the y -th guess that OMEN outputs. This function can simply be computed by running OMEN and printing the probability estimation of the current password. The inverse function $y = f^{-1}(x)$ gives the number of guesses OMEN needs to output before reaching passwords with a certain (estimated) probability. This function is shown in Figure 4 on a double logarithmic scale. In order to simplify the subsequent calculations, we approximate this function as $f^{-1}(x) \approx x^{-1.5}$.

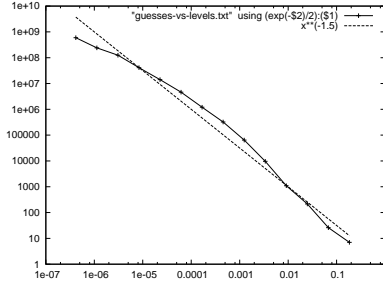


Fig. 4: Relation between (estimated) password probability and the position when it is guessed (line), and the approximation we use (dashed-line), on double-logarithmic scale.

	α	$\ln(\alpha)$	boosted
email	1.6	0.5	0(*)
userName	2	0.7	1
firstName	2.3	0.8	1
lastName	1.5	0.4	0
birthday	5	1.6	2
location	1.7	0.5	1
contact	1.5	0.4	0
eduWork	1.1	0.1	0
friends	1.4	0.3	0
siblings	1.7	0.5	1

Table 3: The estimated values of α and the boosting parameters for the attributes we considered.

The estimated number of guesses which is required to crack *all* passwords pwd_1, \dots, pwd_N is consequently defined by $S = \frac{1}{N} \sum_{i=1, \dots, N} f^{-1}(p_{pwd_i})$. Therefore, for a given $hint_i$, the value α_i to use to boost this hint is the value of α that minimizes the following function: $S^* = \frac{1}{N} \sum_{i=1, \dots, N} (p_{pwd_i}^*)^{-1.5}$.

The following section presents the optimal values of α for different hints.

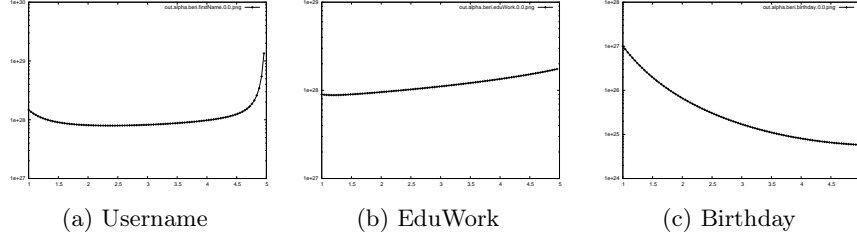


Fig. 6: Estimated influence of the parameter α applied on the attribute: *username* (a), *eduWork* (b) and *birthday* (c). The x -axis shows α and the y axis the expected number of guesses.

4.3 Evaluation

Boosting Parameter Estimation We use the techniques described in the previous section to estimate the boosting parameter α for the different hints. For each hint, we compute the sum S^* for different values of α , and select the value that minimizes it. We illustrate the results with three examples:

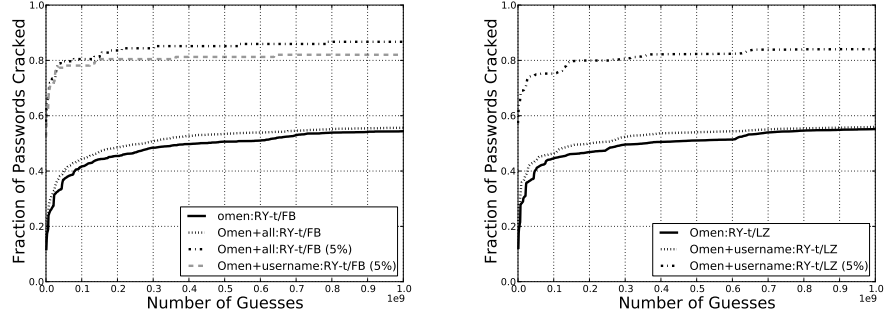
First Name: Figure 6a shows S^* as a function of α for the attribute *first name*. The minimum is around $\alpha = 2.3$, which yields a boosting parameter of 1.

EduWork: Figure 6b shows S^* as a function of α for the attribute *eduWork*, which is an identifier that contains the persons’ education and occupancy. It has a very small decrease in the beginning, with a minimum around $\alpha = 1.2$, but the overall differences are small and the remainder of the graph is monotonically increasing. The boosting parameter is 0.1, which is rounded to 0.

Birthday: Figure 6c shows S^* as a function of α for the *birthday* attribute. Note that our dataset only contains a small number of profiles with that attribute, so the result might not be necessarily meaningful. Overall, there were only 7 profiles where the birthday attribute have an effect, and for two of them the effect is positive. These two have enough effect to lead to a great advantage in using the attribute. Overall, we limited the maximal parameter considered to 5.

Finally, we dropped the attribute *email*. Although it gives an alpha of 1.6 since the *username* is contained in email and achieves better results. All boosting parameters are summarized in Table 5.

OMEN+ Performance Once we have estimated the values of the parameters α_i for each $hint_i$, we run OMEN+ on the Facebook (FB) list, consisting of 3140 passwords together with publicly available information about the users. The results are presented in Figure 7a. As expected, by including personal information in the Markov model, OMEN+ is able to guess more passwords in absolute terms. We have also conducted different experiments with other values α_i to test the effectiveness of our estimation code. We confirmed that, when using different values α_i , the cracking performance either remains the same or slightly decreases. The two lower curves in the figure show the performance of OMEN+ over all passwords with and without using personal attributes. Using personal information can increase the guessed passwords up to 5% (for lower number of guesses



(a) Comparing OMEN with and without (b) Comparing OMEN with usernames as
personal information on the FB list hint and without on the LZ/FB list

of up to 100 million), and around 3% at 1 billion guesses. The limited performance gain is partially explained by the fact that, as shown in Section 4.1, only a small proportion of passwords are based on personal information. The 2 curves in the upper part of the figure display the performance of OMEN+ over the top 5% passwords that are the most correlated with their personal attributes. The achieved performance is much better: About 82% of the passwords are cracked by using usernames only, and more than 88% by using all considered personal attributes. This result is very promising. Figure 7b shows a similar experiment performed on the LZ list (60000 passwords). We only had access to the local-part (i.e., the username) of the email associated to each password. Even though the information in this case was more limited, we realized similar gains compared to the previous test on the more extensive FB data.

5 Discussion and Conclusion

In this work we have first presented an efficient password guesser (OMEN) based on Markov models, which outperforms all publicly available password guessers. For common password lists we found that we can guess almost 70% of the passwords with 10 billion guesses. Subsequently, in our second contribution, we tested if additional personal information about a user can help us to better guess passwords. We found that some attributes indeed help, and we showed how OMEN+ can efficiently leverage this information. We summarize some of the key insights:

- Our work shows that Markov Model have even better potential than previously thought [6], as we could make them guessing “in order”, which leads to the improvements shown in Figures 1a and 1b.
- We find, with our preliminary and simple experiments, that we can guess up to 5% more passwords. However this percentage is a lower bound since we had access to only a limited number of personal information and attributes. Furthermore, we show that the gain can go up to 30% for passwords that are actually based on personal attributes. This result clearly shows that passwords based on personal information are weaker and should be avoided.

References

1. BONNEAU, J., HERLEY, C., VAN OORSCHOT, P. C., AND STAJANO, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. IEEE S&P'12.
2. CASTELLUCCIA, C., DÜRMUTH, M., AND PERITO, D. Adaptive password-strength meters from markov models. NDSS'12.
3. DELL'AMICO, M., MICHARDI, P., AND ROUDIER, Y. Password strength: an empirical analysis. INFOCOM'10.
4. HASHCAT. OCL HashCat-Plus, 2012. <http://hashcat.net/oclhashcat-plus/>.
5. KEDEM, G., AND ISHIHARA, Y. Brute force attack on unix passwords with simd computer. Usenix Sec'99.
6. NARAYANAN, A., AND SHMATIKOV, V. Fast dictionary attacks on passwords using time-space tradeoff. CCS'05.
7. OPENWALL. John the Ripper, 2012. <http://www.openwall.com/john>.
8. PCFG. Matt Weir, 2012. https://sites.google.com/site/reusablesec/Home/password-cracking-tools/probablistic_cracker.
9. PROVOS, N., AND MAZIÈRES, D. A future-adaptive password scheme. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 1999), ATEC '99, USENIX Association, pp. 32–32.
10. WEIR, M., AGGARWAL, S., COLLINS, M., AND STERN, H. Testing metrics for password creation policies by attacking large sets of revealed passwords. CCS 2010.
11. WEIR, M., AGGARWAL, S., DE MEDEIROS, B., AND GLODEK, B. Password cracking using probabilistic context-free grammars. IEEE S&P'09.
12. WORD LIST COLLECTION, 2012. <http://www.outpost9.com/files/WordLists.html>.

Appendix A: Password creation policies

One surprising fact highlighted by the data in the previous section is that usernames and passwords are very similar in a small, yet significant, fraction of the cases. Common sense mandates that a password should not be too similar to the corresponding username, because the username is almost always available to the attacker in a guessing attack.

This fact prompted us to study this specific aspect of password policies in more detail. We conducted a brief test⁸ across 48 popular international sites (from the Alexa Top 500 list), to see how they handle similarities between the username and the password (See Table 4).

These sites have different demands for security, ranging from relatively low security demands (Facebook, Twitter), to high security demands (Ebay, PayPal). We did not rely on the stated password policies, but manually created an account on each site. We created a random but plausible username that was not yet used with the service, and tried to register an account. We initially tried to use the username as the password, and if that failed we tried subsequent modifications until we succeeded. The results are worrisome, but not too surprising. Out of 48 sites tested, 27 allowed *identical* usernames and passwords, including major sites such as Google, Facebook, and Amazon, and only 4 sites required more than one character difference between the two. This could lead to highly effective guessing, for example in the Lulzsec-dataset, we found that 5% of the accounts had strongly overlapping usernames and passwords.

⁸ This survey is neither representative nor complete, but the results are clear enough to show that the problem exists on a large scale.

Account	Username	Password	Same accepted?	Min. Diff. (# Chars)	Comments
Google	berkusrne02@gmail.com	berkusrne02	yes	0	Username cannot be same as email; requires capitals.
Facebook	berkusrne02@gmail.com	berkusrne02	yes	0	
Twitter	berkusrne02	berkusrne03	no	1	
Baidu	berkusrne02	berkusrne03	no	1	
Ebay	berkusrne03	BerkUsrne14	no	2	
Amazon	berkusrne02@gmail.com	berkusrne02	yes	0	
Paypal	berkusrne02@gmail.com	berkusrne03	no	1	
Yahoo	berkusrne02@yahoo.com	berkusrne03	no	1	
Wikipedia	berkusrne02	berkusrne03	no	1	
Windows Live	berkusrne02@hotmail.com	berkusrne03	no	1	
QQ.com	berkusrne02	berkusrne02	yes	0	
LinkedIn	berkusrne02@gmail.com	berkusrne02	yes	0	
Taobao	berkusrne02	berkusrne02	yes	0	
Sina.cn.com	berkusrne02@yahoo.com	berkusrne02	yes	0	
MSN	berkusrne02@gmail.com	berkusrne03	no	1	
WordPress	berkusrne02	berkusrne02	yes	0	
Yandex	berkusrne02	berkusrne03	no	1	
163.com	berkusrne02@163.com	berkusrne03	no	1	
Mail.ru	berkusrne02@Mail.ru	berkusrne03	no	1	
Weibo	berkusrne02@gmail.com	berkusrne02	no	0	
Tumblr	berkusrne02	berkusrne02	yes	0	Password at least 1 capital, 1 number, no 3 consecutive identical characters, not same as account, at least 8 char.
Apple	berkusrne02	BerkUsrne02	no	0	
IMDB	berkusrne02	berkusrne02	yes	0	
Craigslist	berkusrne02@gmail.com	berkusrne03	no	1	
Sohu	berkusrne02@gmail.com	berkusrne02	yes	0	
FC2	berkusrne02	berkusrne03	no	3	
Tudou	berkusrne02@gmail.com	berkusrne02	yes	0	
Ask	berkusrne02	berkusrne02	yes	0	
iFeng	berkusrne02	berkusrne03	no	1	
Youku	berkusrne02	berkusrne02	yes	0	
Tmall	berkusrne02	berkusrne03	no	3	Username length limit.
Imgur	berkusrne02	berkusrne02	yes	0	
Mediafire	berkusrne02@gmail.com	berkusrne02	yes	0	
CNN	berkusrne02	berkusrne02	yes	0	
Adobe	berkusrne02	berkusrne02	yes	0	
Conduit	berkusrne02@gmail.com	berkusrne02	yes	0	
odnoklassniki.ru/	berkusrne02	berkusrne03	no	1	
AOL	berkusrne02	berkusrne03	no	5	
The Pirate Bay	berkusrne	berkusrne	yes	0	
ESPN	berkusrne02	berkusrne02	yes	0	
Alibaba	berkusrne02	berkusrne02	yes	0	Needs capitals or special characters.
Dailymotion	berkusrne02	berkusrne02	yes	0	
Chinaz	berkusrne02	berkusrne02	yes	0	
AVG	berkusrne02@gmail.com	berkusrne02	yes	0	
Ameblo	berkusrne02	berkusrne03	no	1	
GoDaddy	berkusrne02	berkusrne02	yes	0	
StackOverflow	berkusrne02	BerkUsrne03	no	1	
4shared	berkusrne02@gmail.com	berkusrne02	yes	0	

Table 4: Detailed results from a small survey on 48 large sites concerning their password policies.