OMEN: Faster Password Guessing Using an Ordered Markov Enumerator

Markus Dürmuth[1], Fabian Angelstorf[1], Claude Castelluccia[2], and Daniele Perito[2]

[1] Ruhr-University Bochum, Germany
markus.duermuth@rub.de
[2] INRIA, France
{claude.castelluccia | daniele.perito}@inria.fr

Abstract. Passwords are widely used for user authentication, and will likely remain in use in the foreseeable future, despite several weaknesses. One important weakness is that human-generated passwords are far from being random, which makes them susceptible to guessing attacks. Understanding the adversaries capabilities for guessing attacks is a fundamental necessity for estimating their impact and advising countermeasures.
This paper presents OMEN, a new Markov model-based password cracker that extends ideas proposed by Narayanan and Shmatikov (CCS 2005). The main novelty of our tool is that it generates password candidates according to their occurrence probabilities, i.e., it outputs most likely passwords first. As shown by our extensive experiments, OMEN significantly improves guessing speed over existing proposals.
In particular, we compare the performance of OMEN with the Markov mode of John the Ripper, which implements the password indexing function by Narayanan and Shmatikov. OMEN guesses more than 40% of passwords correctly with the first 90 million guesses, while JtR-Markov (for $T = 1$ billion) needs at least eight times as many guesses to reach the same goal, and OMEN guesses more than 80% of passwords correctly at 10 billion guesses, more than all probabilistic password crackers we compared against.

Keywords: Authentication, Password guessing, Markov models

1 Introduction

Password-based authentication is the most widely used form of user authentication, both online and offline. Despite their weaknesses, 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. In fact, while alternative forms of authentication can replace passwords in specific scenarios, they have not been able, so far, to replace them on a large scale [3].

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 resource with which an attacker can generate and verify guesses (e.g., by using specialized hardware and large computing resources [9, 8]), 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.

Tools commonly used for password cracking, such as John the Ripper (JtR) in dictionary mode, 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). This is used to generate new guesses from an existing corpus of data, like a dictionary or a previously leaked password database. Weir et al. [24] demonstrated how to use probabilistic context-free grammars (PCFG) to automatically extract such mangling rules from a corpus of leaked passwords, and Narayanan et al. [16] showed that Markov models, which are known to closely represent natural language, can also be used to guess passwords efficiently. We will demonstrate that, while these attacks already have a good guessing efficiency against passwords, the performance can be substantially improved.

This paper presents OMEN, a new Markov model-based password cracker that generates password candidates according to their occurrence probabilities, i.e., it outputs most likely passwords first. As shown by our extensive experiments, OMEN significantly improves guessing speed over existing proposals.

1.1 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.

In this work we consider *offline guessing attacks*, where an attacker has gained access to this hash and tries to recover the password *pwd*. The hash function is frequently designed for the purpose of slowing down guessing attempts [20]. 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.

John the Ripper: John the Ripper (JtR) [17] 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. Similarly to [6], we discover that for relatively small number of guesses (less than 10^8), JtR in dictionary mode produces best results. In Incremental mode (JtR-inc) [17], JtR tries passwords based on a (modified) 3-gram Markov model.

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 [16], and can likewise be used to accurately estimate the strength of new passwords [5]. Recent independent work [14] compared different forms of probabilistic password models and concluded that Markov models are better suited for estimating password probabilities than probabilistic context-free grammars. The biggest difference to our work is that they only approximate the likelihood of passwords, which does not yield a password guesser which outputs guesses in the correct order, the main contribution of our work.

The underlying idea of Markov models 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 the 2-gram `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 $c_1, \ldots, c_m$, a Markov model estimates its probability as $P(c_1, \ldots, c_m) \approx P(c_1, \ldots, c_{n-1}) \cdot \prod_{i=n}^{m} P(c_i | c_{i-n+1}, \ldots, c_{i-1})$. For password cracking, one basically learns the initial probabilities $P(c_1, \ldots, c_{n-1})$ and the transition probabilities $P(c_n | c_1, \ldots, c_{n-1})$ 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. To make this attack efficient, we need to consider a number of details: Limited data makes learning these probabilities challenging (*data sparseness*) and enumerating the passwords in the *optimal order* is challenging.

Probabilistic Grammars-based Schemes: A scheme based on probabilistic context-free grammars (PCFG) [24] bases on the idea that typical passwords have a certain structure. The likeliness of different structures are extracted from lists of real-world passwords, and these structures are later used to generate password guesses.

Password Strength Estimation: A problem closely related to password guessing is that of *estimating the strength of a password*, which is of central importance for the operator of a site to ensure a certain level of security. In the beginning, password cracking was used to find weak passwords [15]. Since then, much more refined methods have been developed. Typically, so-called pro-active password checkers are used to exclude weak passwords [22, 11, 1, 18, 4]. However, most pro-active password checkers use relatively simple rule-sets to determine password strength, which have been shown to be a rather bad indicator of real-world password strength [23, 12, 5]. The influence of password policies on password strength is studied in [10], and [2] proposes new methods for measuring password strength and applies them to a large corpus or passwords. More recently, Schechter et al. [21] classified password strength by limiting the number of occurrences of a password in the password database. Finally, Markov models have been shown to be a good predictor of password strength while being provably secure [5].

1.2 Paper organization

In Section 2 we describe the *Ordered Markov ENumerator* (OMEN) and provide several experiments for selecting adequate parameters. Section 3 gives details about OMEN's cracking performance, including a comparison with other password guessers. We conclude the paper with a brief discussion in Section 4.

2 OMEN: An Improved Markov Model Password Cracker

In this section we present our implementation of password enumeration algorithm, enumPwd(), based on Markov models. Our implementation improves previous work based on Markov models by Narayanan et al. [16] and JtR [17]. We then present how OMEN, our new password cracker, uses it in practice.

2.1 An Improved Enumeration Algorithm (enumPwd())

Narayanan et al.'s indexing algorithm [16] 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). We developed an algorithm, enumPwd(), 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. [16], according to the formula $lvl_i = \text{round}\left(\log(c_1 \cdot prob_i + c_2)\right)$, 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 the desired number of levels (*nbLevel*), i.e., the levels can take values $0, -1, \ldots, -(\text{nbLevel}-1)$ where *nbLevel* is a parameter. The number of levels influences both the accuracy of the algorithm as well as the running time: more levels means better accuracy, but also a longer running time.

For a specific length ℓ and level η, enumPwd(η, ℓ) proceeds as follows[3]:

1. It computes all vectors $\boldsymbol{a} = (a_2, \ldots, a_\ell)$ of length $\ell - 1$, such that each entry a_i is an integer in the range $[0, \text{nbLevel} - 1]$, and the sum of all elements is η. Note that the vectors have $\ell - 1$ elements as, when using 3-grams, we need $\ell - 2$ transition probabilities and 1 initial probability to determine the probability for a string of length ℓ. For example, the probability of the password "password" of size $\ell = 8$ is computed as follows:

$$P(password) = P(pa)P(s|pa)P(s|as)P(w|ss)P(o|sw)P(r|wo)P(d|or).$$

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

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

```
function enumPwd($\eta, \ell$)
 1. for each vector $(a_i)_{2 \le i \le \ell}$ with $\sum_i a_i = \eta$
    and for each $x_1 x_2 \in \Sigma^2$ with $L(x_1 x_2) = a_2$
    and for each $x_3 \in \Sigma$ with $L(x_3 \mid x_1 x_2) = a_3$
    and for each $x_4 \in \Sigma$ with $L(x_4 \mid x_2 x_3) = a_4$:
    (a) output $x_1 x_2 x_3 x_4$
```

2. For each such vector $\boldsymbol{a}$, 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

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

and transitions have levels

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

– 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.

2.2 The OMEN Algorithm

As presented previously, the enumeration algorithm, enumPwd(η, ℓ) uses two parameters. These two parameter need to be set properly. The selection of ℓ (i.e.

[4] $L(xy)$ and $L(z|xy)$ are the levels of initial and transition probabilities, respectively.

the length of the password to be guessed) is challenging, as the frequency with which a password length appears in the training data is not a good indicator of how often a specific length should be guessed. For example, assume that 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 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 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.

2.3 Selecting parameters

In this section we discuss several parameters choices and examine the necessary trade-off between accuracy and performance. The three central parameters are: n-gram size, alphabet size and the number of levels for enumerating passwords.

n-gram size: The parameter with the greatest impact on accuracy is the size of the n-grams. A larger n generally gives better results as larger n-grams provide a more accurate approximation to the password distribution. However, it implies a larger runtime, as well as larger memory and storage requirements. Note also that the amount of training data is crucial as only a significant amount of data can accurately estimate the parameters (i.e., the initial probabilities and the transition probabilities). We evaluated our algorithm with $n = 2, 3, 4$, results are depicted in Figure 1 (top).

As expected, the larger n is, the better the results are. We have conducted limited experiments with 5-grams, which are not depicted in this graph, but show slightly better results than for 4-grams. As 5-grams require substantially larger running time and memory requirements, for a small performance increase, we decided using $n = 4$.

Alphabet size: The size of the alphabet is another factor that has the potential to substantially influence the characteristics of the attack. Larger alphabet size means that more parameters need to be estimated and that the runtime and memory requirements increase. In the opposite, a small alphabet size means that not all passwords can be generated.

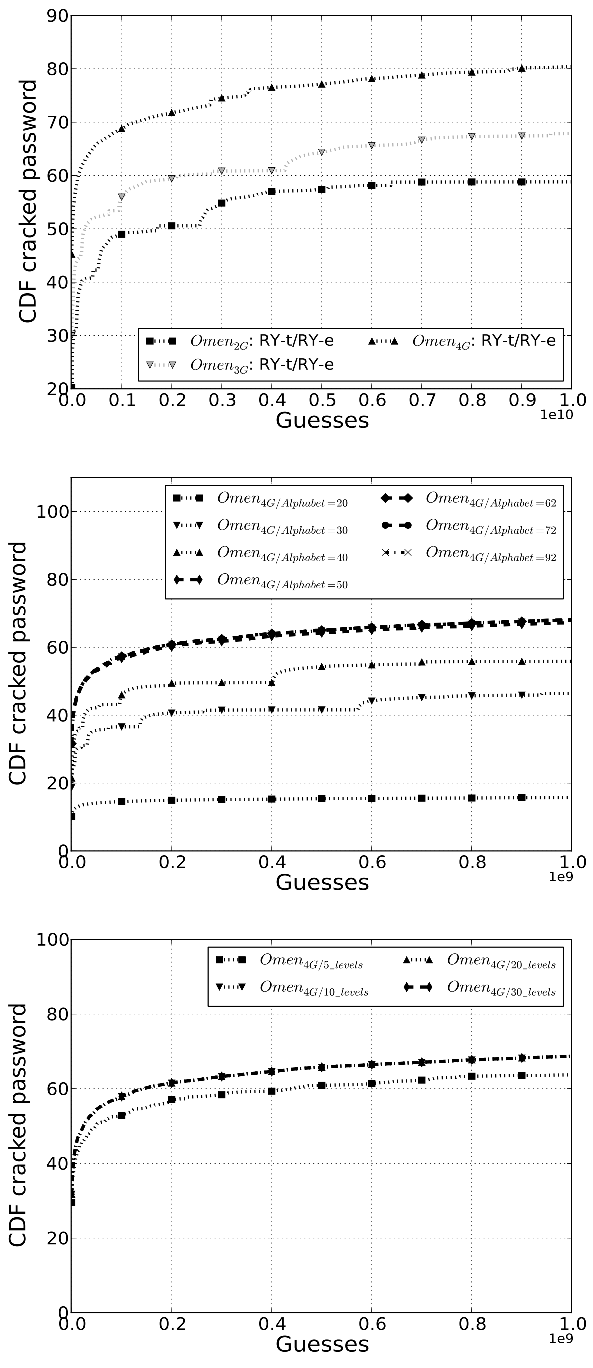

Fig. 1: Comparing different n-gram sizes (top), alphabet sizes (middle), and different number of levels (bottom), for the RockYou dataset.

We tested several alphabet sizes by setting $k = 20, 30, 40, 50, 62, 72, 92$, where the k most frequent characters of the training set form the alphabet. The results are given in Figure 1 (middle). We clearly see an increase in the accuracy from an alphabet size k from 20 to 62. Further increasing k does not noticeable increase the cracking rate. This is mainly explained by the alphabet used in the RockYou dataset where most users favor password with mostly alphanumeric characters rather than using a large number of special characters. To be data independent, we opted for the 72 character alphabet. Note that datasets that use different languages and/or alphabets, such as Chinese Pinyins [13], will have to set different OMEN parameters.

Number of levels: A third important parameter is the number of levels that are used to enumerate password candidates. As for previous parameters, higher number of levels can potentially increase accuracy, but it also increases runtime. The results are shown in Figure 1 (bottom). We see that increasing the number of levels from 5 to 10 substantially increases accuracy, but further increasing to 20 and 30 does not make a significant difference.

Selected parameters: Unless otherwise stated, in the following we use OMEN with 4-grams, an alphabet size of 72, and 10 levels.

3 Evaluating OMEN performance

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

3.1 Datasets

We evaluate the performance of our password guesser on multiple datasets. The largest password list publicly available is the *RockYou list* (RY), consisting of 32.6 million passwords that were obtained by an SQL injection attack in 2009. 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 randomly 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 data cleansing algorithms or leaked from the servers at different points in time). The passwords were obtained in 2006 by a phishing attack.

The *Facebook* list (FB) was posted on the pastebin website (`http://pastebin.com/`) 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.

Ethical Considerations: Studying databases of leaked password has arguably helped the understanding of users real world password practices and as such,

Algorithm	Training Set	#guesses	Testing Set		
			RY-e	MS	FB
Omen	RY-t	10 billion	80.40%	77.06%	66.75%
	RY-t	1 billion	68.7%	64.50%	59.67%
PCFG [24]	RY-t	1 billion	32.63%	51.25%	36.4%
JtR-Markov [16]	RY-t	10 billion	64%	53.19%	61%
	RY-t	1 billion	54.77%	38.57%	49.47%
JtR-Inc	RY-t	10 billion	54%	25.17%	14.8%

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

have been used in numerous studies [24, 23, 5]. Also, these datasets are already available to the public. Nevertheless we treat these lists with the necessary precautions and release aggregated results only that reveal next to no information about the actual passwords (c.f. [7]).

3.2 Comparing OMEN and JtR's Markov Mode

Figure 2 (top) shows the comparison of OMEN and the Markov mode of JtR, which implements the password indexing function by Narayanan et al. [16]. Both models are trained on the RockYou list (RY-t). Then, for JtR-Markov, we fix a target number of guesses T (1 billion or 10 billion), and compute the corresponding level (η) to output T passwords, as required by JtR-Markov.

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 behavior leads to the near-linear curves shown in Figure 2 (top). 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 show the same experiment with T equals to 10 billion guesses (instead of 1 billion). Figure 3 shows the results for 1 billion guesses (left) as well as 10 billion guesses (right), and we see that the linear curve becomes *flatter*.

To show the generality of our approach, we compare the cracking performance on three different datasets: RY-e, FB and MS. The ordering advantage allows OMEN to crack more than 40% of passwords (independently of the dataset) in the first 90 million guesses while JtR-Markov cracker needs at least eight times as many guesses to reach the same goal. For the RockYou dataset the results most pronounced: For instance, OMEN cracks 45.2% of RY-e passwords in the first 10 million guesses (see Figure 3 (right)) while JtR-Markov achieves this result after more than 7 billion guesses (for $T = 10$ billion).

In the above comparison, OMEN uses 4-grams (c.f. Section 2.3), while JtR-Markov uses 2-grams. To see the effects that this difference has, we provide an additional comparison of OMEN using 2-grams with JtR-Markov, this is given in Figure 4. The results are as expected: JtR-Markov still gives a straight line, which

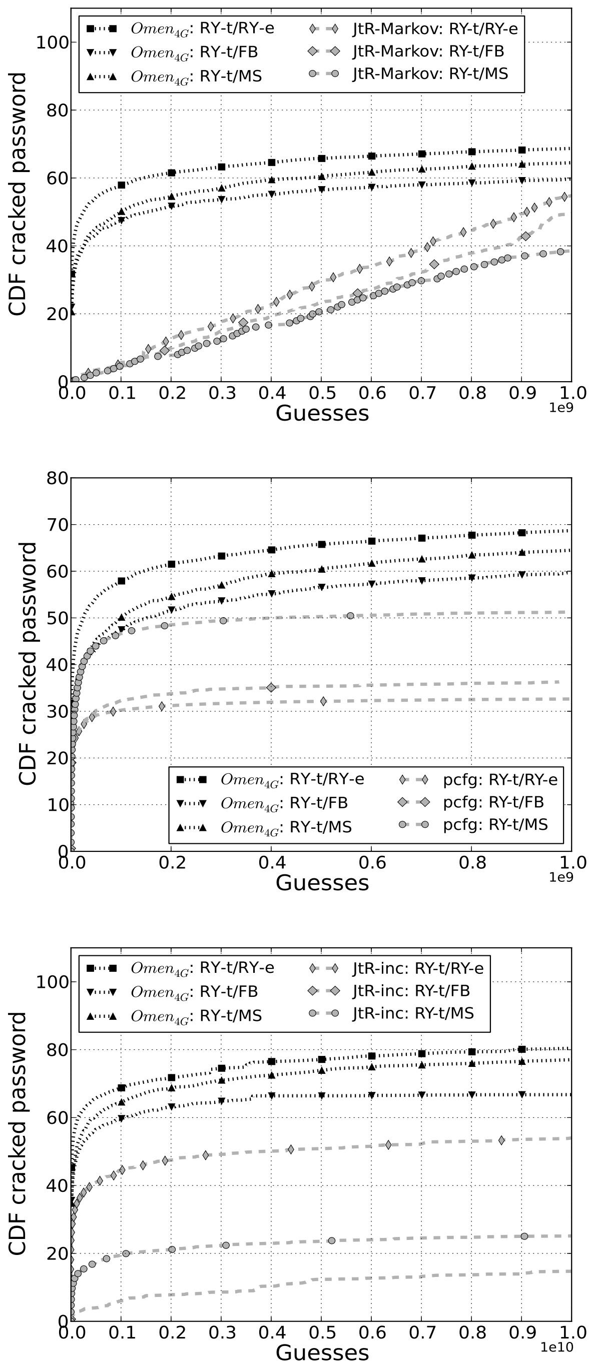

Fig. 2: Comparing OMEN with the JtR Markov mode at 1B guesses (top), with the PCFG guesser (middle), and with JtR incremental mode (bottom).

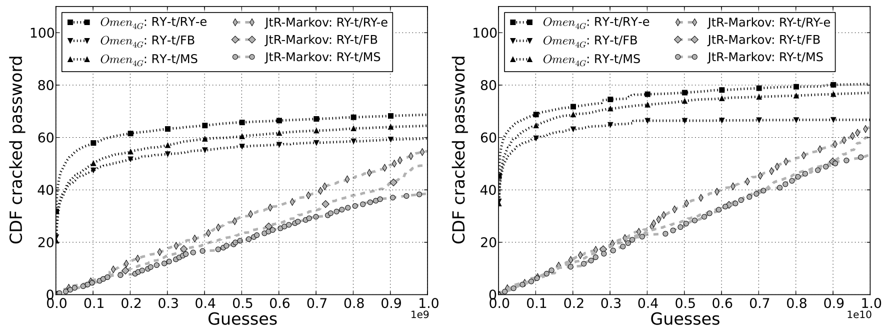

Fig. 3: Comparing OMEN with the JtR Markov mode at 1 billion guesses (left), and at 10 billion guesses (right).

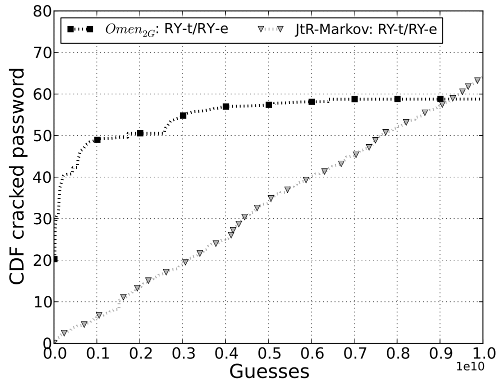

Fig. 4: Comparing OMEN using 2-grams with JtR Markov mode.

means that OMEN has a better cracking speed. The speed advantage of OMEN can be seen at 1 billion guesses where OMEN cracks 50% of all passwords while JtR-markov cracks less than 10%. At the point T, i.e., when JtR-Markov stops, both algorithms perform roughly the same. Note that since not all parameters (i.e., alphabet size, number of levels etc.) of both models are identical, we have a small difference in the cracking rate at the point T.

3.3 Comparing OMEN and PCFG

Figure 2 (middle) compares OMEN to the PCFG password guesser of Weir et al. [24], based on the code available at [19]. We run it using the configuration as described in the paper: we use RY-t to extract the grammar and the dictionary dict-0294 [25] to generate candidate passwords.

Figure 2 shows that OMEN outperforms the PCFG guesser. After 0.2 billion guesses, OMEN cracks 20% more passwords than PCFG for both RY-e and FB datasets and 10% more for MS. It is interesting to see the impact of the training set on PCFG performance: PCFG performs much better on MS than on FB and RY-e. We believe the reason is that the grammar for PCFG is trained on a subset of the MS list, and thus the approach is better adapted for guessing passwords from the MS list. OMEN achieves roughly the same results for all datasets which proofs the robustness of the learning phase. Finally, note that PCFG mostly plateaus after 0.3 billion guesses and results hardly improve any more, whereas OMEN still produces noticeable progress.

3.4 Comparing OMEN and JtR's Incremental Mode

We also compare OMEN to JtR in incremental mode, see Figure 2 (bottom). Similarly to the previous experiments, both crackers were trained on the RockYou training set of 30 million passwords and tested on RY-e, MS and FB datasets. Clearly, JtR incremental mode produces worse guesses than OMEN.

4 Discussion and Conclusion

In this work, we have 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 more than 80% of passwords with 10 billion guesses. While Markov models were known [16] to be an effective tool in password guessing, previous work was only able to output the corresponding guesses in an order dictated by algorithm internals (and pretty much unrelated to their real frequency), OMEN can output guesses in order of (approximate) decreasing frequency and thus dramatically improves real-world guessing speed. Moreover, we performed a number of experiments to assess the impact of different parameters on the accuracy of the algorithm and find optimal parameters. We believe that OMEN can be useful as a preventive measure by organizations to verify that their members do not select "weak" passwords.

References

1. M. Bishop and D. V. Klein. Improving system security via proactive password checking. *Computers & Security*, 14(3):233–249, 1995.
2. J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Proc. IEEE Symposium on Security and Privacy*. IEEE, 2012.
3. J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proc. IEEE Symposium on Security and Privacy*. IEEE, 2012.
4. W. E. Burr, D. F. Dodson, and W. T. Polk. Electronic authentication guideline: NIST special publication 800-63, 2006.
5. C. Castelluccia, M. Dürmuth, and D. Perito. Adaptive password-strength meters from Markov models. In *Proc. Network and Distributed Systems Security Symposium (NDSS)*. The Internet Society, 2012.
6. M. Dell'Amico, P. Michiardi, and Y. Roudier. Password strength: an empirical analysis. In *Proc. 29th conference on Information communications*, INFOCOM'10, pages 983–991, Piscataway, NJ, USA, 2010. IEEE Press.
7. S. Egelman, J. Bonneau, S. Chiasson, D. Dittrich, and S. Schechter. It's not stealing if you need it: A panel on the ethics of performing research using public data of illicit origin. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2012.
8. HashCat. OCL HashCat-Plus, 2012. `http://hashcat.net/oclhashcat-plus/`.
9. G. Kedem and Y. Ishihara. Brute force attack on unix passwords with SIMD computer. In *Proc. 8th conference on USENIX Security Symposium - Volume 8*, SSYM'99. USENIX Association, 1999.
10. P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proc. IEEE Symposium on Security and Privacy*. IEEE, 2012.
11. D. V. Klein. Foiling the cracker: A survey of, and improvements to, password security. In *Proc. USENIX UNIX Security Workshop*, 1990.
12. S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *CHI 2011: Conference on Human Factors in Computing Systems*, 2011.
13. Z. Li, W. Han, and W. Xu. A large-scale empirical analysis of chinese web passwords. In *Proc. 23rd USENIX Security Symposium (USENIX Security)*, Aug. 2014.
14. J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *Proc. IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2014.
15. R. Morris and K. Thompson. Password security: a case history. *Communications. ACM*, 22(11):594 – 597, 1979.
16. A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proc. 12th ACM conference on Computer and communications security (CCS)*, pages 364–372. ACM, 2005.
17. OpenWall. John the Ripper, 2012. `http://www.openwall.com/john`.
18. The password meter. Online at `http://www.passwordmeter.com/`.
19. PCFG Password Cracker implementation. Matt Weir, 2012. `https://sites.google.com/site/reusablesec/Home/password-cracking-tools/probablistic_cracker`.

20. N. Provos and D. Mazières. A future-adaptive password scheme. In *Proc. Annual conference on USENIX Annual Technical Conference*, ATEC '99. USENIX Association, 1999.
21. S. Schechter, C. Herley, and M. Mitzenmacher. Popularity is everything: a new approach to protecting passwords from statistical-guessing attacks. In *Proc. 5th USENIX conference on Hot topics in security*, pages 1–8. USENIX Association, 2010.
22. E. H. Spafford. Observing reusable password choices. In *Proc. 3rd Security Symposium*, pages 299–312. USENIX, 1992.
23. M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. 17th ACM conference on Computer and communications security (CCS 2010)*, pages 162–175. ACM, 2010.
24. M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *Proc. IEEE Symposium on Security and Privacy*, pages 391–405. IEEE Computer Society, 2009.
25. Word list Collection, 2012. `http://www.outpost9.com/files/WordLists.html`.