

Local Password Validation Using Self-Organizing Maps

Diogo Mónica and Carlos Ribeiro

INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa

Rua Alves Redol 9, 1000-029, Lisboa

{diogo.monica, carlos.ribeiro}@ist.utl.pt

<http://www.gsd.inesc-id.pt>

Abstract. The commonly used heuristics to promote password strength (e.g. minimum length, forceful use of alphanumeric characters, etc) have been found considerably ineffective and, what is worst, often counterproductive. When coupled with the predominancy of dictionary based attacks and leaks of large password data sets, this situation has led, in later years, to the idea that the most useful criterion on which to classify the strength of a candidate password, is the frequency with which it has appeared in the past.

Maintaining an updated and representative record of past password choices does, however, require the processing and storage of high volumes of data, making the schemes thus far proposed centralized. Unfortunately, requiring that users submit their chosen candidate passwords to a central engine for validation may have security implications and does not allow offline password generation. Another major limitation of the currently proposed systems is the lack of generalisation capability: a password similar to a common password is usually considered safe.

In this article, we propose an algorithm which addresses both limitations. It is designed for local operation, avoiding the need to disclose candidate passwords, and is focused on generalisation, recognizing as dangerous not only frequently occurring passwords, but also candidates similar to them. An implementation of this algorithm is released in the form of a Google Chrome browser extension.

Keywords: password validation, dictionary attacks, self-organizing maps.

1 Introduction

The need to promote the use of strong passwords has led to the widespread use of password validation heuristic rules, (e.g. minimum length, forceful use of alphanumeric characters, etc). However, these rules are largely ineffective (e.g. "p@ssw0rd" will typically be considered a sufficiently strong password) and are often counterproductive. Minimum size requirements, for example, will typically favor the choice of plain text phrases or other low entropy sequences [1], since users want to be able to remember the chosen long password strings. Even though possibly strong in terms of brute force attacks, this undesirable side-effect leaves the chosen passwords highly vulnerable to dictionary attacks. Also, these heuristics are inadvertently leading users to the repeated use of very common solutions, which can be easily remembered, while still obeying

the requirements. In fact, users typically circumvent the imposed pseudo-randomness, low-memorability, by using a few common tricks ("p@ssw0rd" being again a typical example), which are then used repeatedly. This leads to the reiterated use of what are, in fact, very weak passwords, highly vulnerable to statistical guessing attacks (a dictionary based attack ordered by decreasing probability of occurrence). When coupled with the predominancy of dictionary based attacks, these password validation rules may therefore be decreasing overall security levels, instead of promoting them.

This situation has led, in later years, to the idea that the most useful criterion on which to classify the strength of a given password, is the frequency with which it has appeared in the past (e.g. [2], [3], [4]). In fact, some organizations, like Twitter, are already prohibiting further use of their most common passwords, to hamper the effectiveness of statistical dictionary based attacks.

Maintaining an Internet-wide updated and representative record of past password choices does, however, require access to the passwords of at least some representative internet-scale systems (with a high number of users) and the processing of high volumes of data. The schemes thus far proposed are, therefore, centralized. When users want to verify the adequacy of a given password, they will typically submit the password to this centralized engine for validation. While such centralized validation may be acceptable when effected within the scope of the organization under which the password is to be used, it creates some security concerns if users want to validate passwords in a generic engine, since it implies the disclosure of the candidate password. Also, it requires connectivity whenever password choices are to be made.

Alternatively, a compressed version of the observed password database may be downloaded, to enable local validation. This shift towards local validation avoids the need to disclose candidate passwords but, due to the huge amount of passwords (tens to hundreds of millions) in any representative snapshot of past history, requires rates of compression which lossless compression cannot achieve. Another fundamental concern with the local validation approach is the need to guarantee that diffusion of the validation database does not compromise existing passwords, by publicly distributing information capable of supporting dictionary based attacks.

A viable (lossy) compression mechanism was proposed in [4], where the authors propose the creation of an oracle to identify undesirably popular passwords. Their scheme is based on an existing data structure known as a *count-min sketch* (CM-sketch), which the authors populate with existing users' passwords and use to efficiently track password popularity within the user base. The solution proposed addresses both the compression and the password compromise concerns but does so in a centralized fashion.

However, independently of the underlying implementation details of such a popularity-based password validation scheme (local or centralized, compression rates, etc), the overall concept has an underlying weakness, which stems from the way humans tend to make their password choices: when faced with the rejection of their chosen password, users will, with high probability, search for minor variations "close" to the desired candidate password, by introducing or replacing non-defacing elements (and thereby maintaining its memorability) until an acceptable instance is obtained. This behavior is the reason passwords such as "p@ssword", "ashley123", or "il0veu" have become so common.

Since attackers are well aware of this type of behavior, smarter dictionary attacks already explore the “*neighborhood*” of the historically more frequent passwords before moving to the next one. In fact, many publicly available password recovery tools are already using several different sets of rules to produce variations of the known dictionary entries (e.g. Hashcat’s [5] “best64.rules”). For password validation, this means that if, for example, “password1” is historically a very frequent choice, then “*close-by*” variations such as “password2”, or “p@ssw0rd1” should also be classified as weak, even if historically their frequency would not classify them as such.

The goal of this paper is therefore to design a popularity based classification scheme, which is not only capable of classifying a password as weak because it is historically a frequent choice, but also because it is “*too close*” to a frequent password and, hence, is likely to be guessed by a modern statistical attack. Another important side effect of this generalisation capability is the fact that it allows local validation without the need for frequent updates of the validation database, since this intrinsic generalisation capability will protect users from the evolutionary changes of the database between updates. The proposed scheme is envisaged for local operation and, as such, addresses all the above discussed issues concerning size, connectivity, and non-disclosure of historical passwords. As a proof-of-concept we released an implementation of this scheme in the form of a Google Chrome browser extension that anyone can obtain for free on the chrome web store.

The rest of this paper is organized as follows: Section 2 describes the proposed approach, and discusses individually its constituting steps. Section 3 evaluates the performance of the proposed scheme. Section 4 details our proof-of-concept implementation. Finally, Section 5 concludes the paper, and discusses directions for future work.

2 Our Approach

There are three conceptual steps in our classification engine: *compression*, *generalisation* and *hashing*. When validating a password, users will execute the fourth and final conceptual step of the overall scheme: *classification*. This last step is the only that the final user is required to do; it is also the only step where algorithm execution speed is critical. The overall proposed algorithm is depicted in Figure 1, and is best addressed by discussing the *compression* step first.

2.1 Compression

Let us discuss the compression step as applied directly to the raw passwords. The possibility of pre-processing prior to compression (e.g. feature extraction) will not be addressed in this paper.

Since the overall algorithm must recognize the concept of *closeness* between passwords, the compression step must preserve the topological proximity of the input passwords. That is: passwords which are close in the input (non-compressed) space should be also close in the output (compressed) space. This precludes the use of algorithms such as the CM-sketch presented in [4]. In this article, we will be using Self-Organizing Maps (SOM) to achieve the intended similarity preserving compression. SOM are a

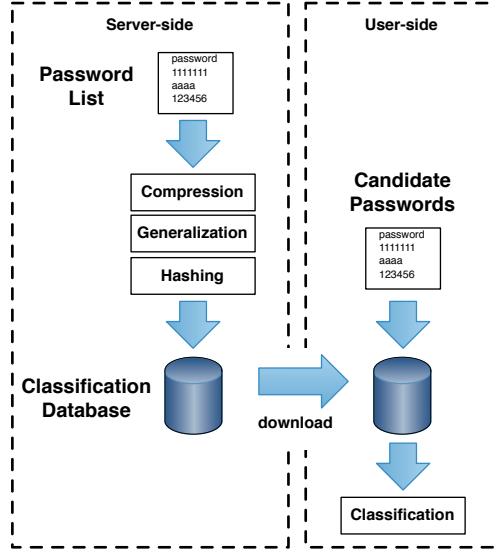


Fig. 1. Overall architecture

simple, well known clustering tool, capable of providing very synthetic summaries of the input data. They are a type of unsupervised neural network, capable of reflecting in the output space the topological proximity relations present in the input space. As will be seen, using a SOM for the compression step of our overall algorithm also provides an easy way to achieve the desired generalisation capability.

Self-Organizing Maps (SOM). A SOM is a lattice of nodes (a 2D rectangular lattice is used in this article). Each node has both a topological position within the lattice (x, y coordinates, in our case), and a vector of weights (the model, m) of the same dimension of the input vectors. Typically, the model vectors are randomly initialized. Training occurs over many iterations; at each iteration, the input vectors (previously used passwords, in our case) are presented to the network, one at a time, in random order (training can also be done in batch, but that is merely an implementation issue). For each input password, the following steps are executed:

- Determine which node has a model *closer* to the input password (according to the similarity measure chosen for the input space). This node is commonly called the BMU (Best Matching Unit);

$$i_{BMU} = \min_i \{ \text{similarity}(\text{node}_i, \text{password}) \}$$

- Find the set \mathcal{N} of all nodes in the lattice neighborhood of the BMU. The radius r of this neighborhood is typically monotonically decreasing; in the first iterations, large

radii are used for the definition of the neighborhood, but the radius will decrease for later iterations;

$$\mathcal{N} = \{node_i : \sqrt{(x_i - x_{BMU})^2 + (y_i - y_{BMU})^2} \leq r\}$$

- Update the models of all nodes in the lattice neighborhood of the BMU, to make them approximate the input password. Typically, the closer a node is to the BMU in the lattice, the more its model is changed towards the input password, according to some monotonically decreasing function $h(\delta x_i, \delta y_i)$, δx_i and δy_i being the difference in lattice coordinates between node i and the BMU:

$$m_i(t+1) = m_i(t) + h(\delta x_i, \delta y_i) \alpha (password_t - m_i(t)).$$

In this equation, α , the *learning factor* ($0 < \alpha < 1$), determines how heavily the models should be pulled towards the input password at each iteration (α is, typically, a decreasing function of the iteration number).

Even though several minor variations have been proposed and analysed in the literature, particularly in what concerns the shape and behavior of r , α and $h(\delta x, \delta y)$ as training proceeds (these details will not be discussed here, since they are essentially not relevant for our results), the intended result of the training step is always the same: to have the nodes of the latticed pulled towards the input vectors, but preserving the desired degree of neighborhood cohesion. Since the number of nodes in the lattice is much smaller than the number of input passwords, the resulting map is a summary replica of the input space, with a much lower number of elements, but maintaining its topological relations. The literature on SOM is abundant. For further details, see, for example, [6] and [7].

Compression Ratio. The compression ratio is $\frac{N}{M}$, N being the number of input passwords, and M the number of nodes in the SOM. As will be shown later, with the proposed implementation, $p_{miss} = 0$, for all compression ratios. That is: for any chosen compression ratio, there will be a null probability of failing to flag as dangerous a password whose historical frequency of appearance is higher than the defined threshold. This means that one may choose any desired rate of compression without fearing that excessive compression may provoke an incorrect password validation. However, if the compression ratio is chosen too high, there will be a loss in the discriminating capability of the network, since too many input passwords will be mapped into each node/neighborhood, resulting in a high false alarm rate when validating prospective candidates. As will be seen in Section 3, good results have been obtained for maps of a few thousand nodes (e.g. 6000 nodes), even when summarising lists with tens of millions of passwords.

Similarity Measures. The concept of similarity is central to SOM operation, since it defines the topological characteristics of the input space to be preserved in the output space. In this particular application, we are therefore faced with the issue of defining an appropriate measure of similarity between passwords. This is, in fact, a two-pronged

problem. Firstly, the similarity measure to be used must reflect in some appropriate manner the user perception of "closeness" between passwords, something which is clearly not uniquely defined. Many distances and/or similarity measures for categorical data have been proposed (e.g. [8], [9]), but their resulting notion of similarity is not always compatible with the type of minor variations that a user is likely to use when forced to modify a chosen (and rejected) password. Secondly, the chosen measure of similarity must allow the concept of "fractional approximation". When training the SOM, we will need to change the node models, to make them approximate the input password, and there are no unique solutions to the problem of approximating categorical data; also, we will want to use diminishing degrees of approximation as the distance to the BMU increases, as determined by $h(\delta x, \delta y)$; this implies the use of a similarity measure for which an increase in similarity can be defined in a quasi-continuous way. In this article, we will be using a mixed, non-pure, measure of similarity, capable of addressing both problems. Designating by $d(p_1, p_2)$ the dissimilarity between passwords p_1 and p_2 , we define.

$$d('password', 'p@ssw0rd') = \sqrt{(97 - 64)^2 + (111 - 48)^2} \cdot 2^2 = 284.48, \quad (1)$$

$$d(p_1, p_2) = \sqrt{\sum_{j=1}^n (p_1(j) - p_2(j))^2 \cdot \text{hamm}(p_1, p_2)^\beta}, \quad (2)$$

where n is the maximum number of characters of the input passwords, $p_1(j)$ is the ASCII code of the j^{th} character of p_1 (same for $p_2(j)$), and $\text{hamm}(p_1, p_2)$ is the character-wise Hamming distance between the two passwords (that is, the number of characters in which they differ), often referred to as the "overlap measure" (e.g. [8]). For example, for $\beta = 2$, the dissimilarity between "password" and "p@ssw0rd", according to this measure, will be 284.48, as given by the product of 71.12 (euclidean distance between ASCII codes) and 4 (square of the character-wise Hamming distance: 2). Two comments should be made concerning this dissimilarity measure:

- Firstly, we note that the first factor allows for a quasi-continuous solution to the approximation problem: approximating a model from the input password will be made by simply approximating the corresponding ASCII codes. The j^{th} character of the i^{th} model will be updated as:

$$\forall j : m_{i,j}(t+1) = m_{i,j}(t) + \Delta, \quad (3)$$

where

$$\Delta = \text{round}(h(\delta x_i, \delta y_i) \alpha(p_t(j) - m_{i,j}(t))).$$

The approximation of the ASCII codes will also generate, at discrete steps, a decrease in the Hamming distance between the model and the input password and, thus, a decrease in the second factor;

- Secondly, we should note the role of the β parameter. Since the Hamming distance bears a stronger correlation with the human concept of closeness between

passwords, an increase in β will pull the overall measure of dissimilarity towards a simple human-related overlap distance, desensitizing the dissimilarity measures relative to the numerical distance between the ASCII codes. In this article, we will use $\beta = 2$.

Procedure. The SOM models are typically initialized with random values between 0 and 128. Training is done with as few as 30/40 iterations, with monotonically decreasing values of the learning factor α and neighborhood radii. The SOM models are updated as presented in Section 2.1. Once training is completed, the full list of passwords is again presented to the network, and nodes are labeled with the (absolute) frequency of appearance of the passwords for which that node is the BMU. If more than one password maps to the same node, the node's label is the frequency of appearance of the most frequent one. The label on each node is, therefore, a measure of the popularity of the most frequent password on the training set for which that node is the BMU.

2.2 Generalisation

Once the compression phase completes, we are left with a map which could be directly used for password classification, as follows: i) a popularity threshold is chosen; ii) each new candidate password is presented to the network, and the corresponding BMU determined; iii) if the popularity level of the resulting BMU is above the chosen threshold, the candidate password is rejected, due to similarity with existing popular passwords; otherwise, it is accepted as a valid password.

Note that, by design, any password in the training set more popular than the chosen threshold will map to a BMU with a popularity label greater or equal to the chosen threshold. Denoting by $L(i)$ the popularity label of the i^{th} node, by $BMU(p)$ the BMU corresponding to password p , and by $f(p)$ its frequency of appearance, we thus have that:

$$\forall \tau : f(p) \geq \tau \Rightarrow L(BMU(p)) \geq \tau. \quad (4)$$

This means that the probability of wrongly classifying a password whose occurrence is higher than the threshold as safe (p_{miss}) is 0, as intended.

At this point, the generalisation capabilities of the scheme may be smaller than desired. At the output of the compression phase, it is possible to have nodes with very low popularity adjacent to nodes of high popularity. However, the topological preservation property of the SOM implies that, with high probability, their models are close to one another. Hence, if one of them is very popular as a BMU within the training data, the desired generalisation capability would dictate that nearby models should also be avoided, even if they were not popular within the password list used to train the SOM.

The generalisation capability of the network may thus be increased by imposing some popularity leakage from local maxima to neighboring nodes. This can be achieved by simple spacial low-pass filtering of the popularity levels across the map. Let us designate by $\phi(x, y)$ the popularity label of the node with lattice coordinates (x, y) . The smoothed version of the SOM (and the desired level of generalisation capability) can be obtained as in (5):

$$\phi(x, y) * K(x, y), \quad (5)$$

where the $*$ operator stands for 2D convolution, and $K(x, y)$ is any chosen low pass kernel. Since $\phi(x, y) \geq 0$, to preserve the $p_{miss} = 0$ property, we need to condition the chosen kernel to $K(0, 0) \geq 1$. In this article, no attempt to optimize the choice of the kernel was made. Smoothing was performed with a simple non-linear 2D lowpass mask, designed to preserve local maxima and their popularity levels:

$$K(x, y) = \max(k_1(x, y), k_2(x, y)), \quad (6)$$

$k_1(x, y), k_2(x, y)$ being the 2D kernels in Figure 2.

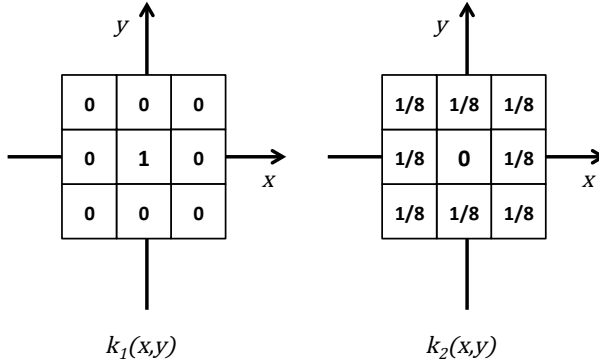


Fig. 2. Smoothing kernel

Decreasing the bandwidth of the used kernel will increase the generalisation capability. However, it will concurrently decrease the resolution of the classification scheme, since the resulting popularity leakage may force bigger subsets of the password space to be flagged as "too popular" and, thus, become unusable.

2.3 Hashing

At the end of the generalisation phase, the SOM map is finalized, and ready to classify candidate passwords as "acceptable" or "not acceptable", depending on the popularity label of the corresponding BMU. However, at this point, the models in the SOM are a compressed summary of the training passwords. As such, the SOM presents a security risk, since it carries too much information concerning these passwords. In fact, given a strong enough preponderance of the most common training passwords, some models may constitute pure copies of those passwords. This potential security problem becomes even more severe in our case, where a distribution of the SOM is envisaged, to allow client-side offline operation.

To address this problem, the SOM resulting from the generalisation step is hashed, prior to storage and/or disclosure. On the user side, classification will therefore be made on hashed space. As in the compression step, the hash function must not, however, destroy the topological proximity relation between the input and output spaces. If a candidate password is similar to a popular one, their hashes must also be similar. Otherwise,

they would not deterministically map to neighboring BMUs, and the intended generalisation capability would be lost.

There is a considerable body of work on similarity preserving hash functions (e.g. [10] to [11]). However, in this particular application, two additional constraints must be met by the hash functions:

- Computing the hash for a single password must not require knowledge of all other passwords. Otherwise, users will not be capable of hashing their candidate passwords;
- The hash function must not be invertible. Otherwise, possession of the hashed SOM would still constitute an attack vector for password estimation.

These two conditions imply that previously proposed similarity preserving hashing methods cannot be used.

The first condition implies that alternatives relying on computation of the adjacency matrices between all passwords (such as in e.g. [11]) cannot be used, since the set of known passwords will not be available to the user when hashing candidate passwords.

The second condition poses a more general difficulty. Cryptographic hashes clearly comply with this condition, but are meant, by design and definition, to destroy any topological proximity relations existing in the input data; they cannot, therefore, be directly used in this context. Similarity preserving hashes such as e.g. [10] or [12], on the other hand, are capable of maintaining proximity relations (up to a point), but were never designed to be non-invertible. The hash functions used in the frequently used Locality Preserving Hashing methods ([10], [13] and related work) are deterministically invertible; all methods based on projection on random-spaces such as, for example, the one found in [12], are prone to inversion attacks if the projection matrix and shift vector are known (and they must be known to users, since they must be capable of hashing their candidate passwords). As pointed out in [12], if these quantities are unknown, the method would be secure, from an informational point of view, but that is not the case here. Forcing the non-invertibility of the projection matrix used on these methods does also not provide enough security. Reconstruction can still be attempted and attack vectors obtained algebraically (i.e. by careful use of the Moore-Penrose pseudo inverse, or similar techniques).

In this paper, hashing will still be made by linear vector projection, to maintain proximity relations. The possibility of reconstruction is avoided by suppressing some of the information concerning the resulting vectors. More precisely, passwords are projected into the Fourier harmonic space, and the resulting phase information is discarded. That is:

$$\text{hash}(p) = |\text{FFT}(p)| \quad (7)$$

The classification database to be sent to users is, therefore, not the trained SOM, but a map of the magnitude of the Fourier transform of the nodes' models. The absence of phase information means that the models cannot be reconstructed, and, therefore, the set of training passwords is not compromised by diffusion of the classification database. In fact, the absolute values of the Fourier transform of the models only contain information concerning their second moments (as directly results from the Wiener-Kintchin theorem). Passwords cannot thus be inferred from the classification database (even though their autocorrelation function can).

One consequence of the informational loss in the hashing step is that, when classifying a candidate password, users will not be evaluating its direct proximity to the nodes of the trained SOM, but will, instead, be evaluating their spectral similarity. This does not affect the $p_{miss} = 0$ perfect performance (the power spectrum of a password will always be closer to itself than to the power spectrum of a different password), but will introduce a slightly different measure of similarity in the classification phase. Namely, the Hamming distance between the candidate password and the nodes of the SOM cannot be considered at this point, since there simply is not enough information to compute such a distance. This also means that, after the hashing step, the popularity levels of the nodes must be recomputed, since some of the mapping of passwords to nodes in the spectral domain may differ from the corresponding mappings before transformation.

Two further notes must still be made, before addressing the final step of the algorithm. Firstly, we note that, since all models are real valued, their Fourier transforms are symmetric around the origin and, hence, the magnitude of the negative frequency bins can be discarded, thus reducing the size of the classification database by almost a factor of 2. Also, to further restrict the amount of information present in the distributed classification database, the nodes' popularity labels can be recoded at this point with a single bit, to simply label nodes as being "on or above danger threshold" (e.g. "1") or "below danger threshold" (e.g. "0"), since that is all the information needed for the intended binary classification. However, sometimes, the user may want to have some control on the classification threshold being used. A more stringent threshold may be desired in critical contexts (e.g. bank accounts' passwords) than in more relaxed situations (e.g. temporary accounts). Therefore, the popularity levels of the models may be coded with more bits, corresponding to any desired number of discrete levels. The user will then be capable of selecting the desired "password strength" level, with passwords being classified as "dangerous" only if their corresponding BMU has a popularity label higher than the level selected by the user as threshold.

An example of a map of 6000 nodes trained with the Rockyou password set is shown in Figure 3. Forbidden nodes (above threshold) are represented in black and acceptable nodes in white. We also represent the same map after smoothing (with the smoothing kernel of Figure 2) in grey color.

2.4 Classification

The classification step is now trivial. The classifying tool made available to users simply:

1. Computes the power spectrum of the candidate password;
2. Using the local copy of the classification database, determines the BMU (node with the lower Euclidean distance to the power spectrum of the candidate password);
3. Rejects the candidate if the label of the BMU is "1", and validates it as an acceptable password if the BMU's label is "0"

We note that this classification step is very fast. All the computational effort of the overall method lies in the training phase, which is centrally made by the classification database provider, making this method ideal for password validation even in resource constrained devices (e.g. mobile devices).

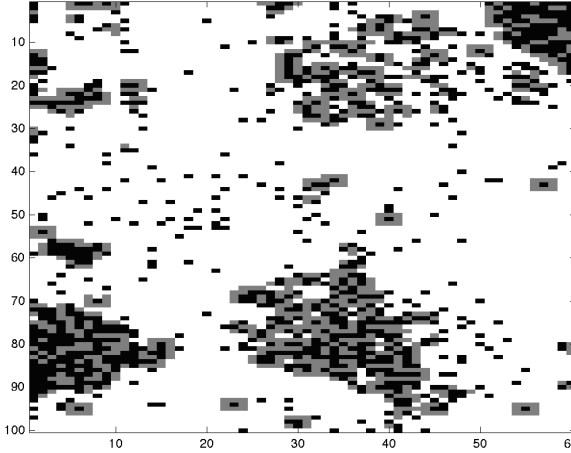


Fig. 3. SOM Map with 6000 nodes

3 Performance

To evaluate the performance of the proposed scheme, we will need to consider its i) *compression rate*, ii) *statistical performance*, and iii) *generalisation capability and consistency*. As expected, these three indicators are highly interdependent (e.g. higher compression rates will typically imply inferior statistical performance). For the first two indicators (compression rate and statistical performance), we will be able to compare the proposed scheme with the CM-sketch [4]. The third parameter must, however, be evaluated without the benefit of comparison since, to our knowledge, there is no other proposed scheme for password validation which, while operating based on the popularity levels of previously seen passwords, is capable of generalizing and flagging as dangerous passwords which, even though not previously seen, are too close to popular choices. A final relevant parameter to be discussed is iv) *classification speed*.

3.1 Compression Rate and Statistical Performance

Since the intended use of the proposed scheme involves a user downloading the classification database, it is important to evaluate the achievable compression rates, and the decrease in performance with increasing compression rate. Let us compare the performance of the SOM approach with CM-sketch[4] by looking at the passwords in the Rockyou set [14] (32602348 passwords of that list were used; the maximum password size is 30 bytes).

Firstly, we must note that both approaches will have $p_{miss} = 0$, for any given compression rate. That is: any password belonging to the training set with a popularity level above the defined threshold will be flagged as dangerous with unit probability, independently of the size of the sketch or the SOM map and, hence, the obtained compression rate. Hence, comparison will be made from the point of view of p_{FA} , the probability

of false alarm. The theoretical bounds for the CM-sketch are given in [15]: for a (d, w) CM-sketch, the required number of lines d (each line corresponds to a different hash function) and columns w can be obtained as follows:

$$w = \left\lceil \frac{e}{\epsilon} \right\rceil, \quad d = \left\lceil \ln \frac{1}{\delta} \right\rceil, \quad (8)$$

where e is Napier's constant, and $(1 - \delta)$ is the probability that the estimated number of occurrences of any given password (\hat{n}_i) will be within a semi-interval of ϵN of the real number of occurrences (n_i):

$$\hat{n}_i \leq n_i + \epsilon N, \quad (9)$$

N being the total number of passwords. However, the typical performance of the CM-sketch will typically be well within the theoretical bounds. As such, using these bounds would present the CM-sketch case in an unduly unfavorable light. Furthermore, no equivalent analytic bounds exist for the SOM case. As such, the comparison will be made by Monte Carlo simulation. The SOM models were initialized with random values between 0 and 128. Training was done with a few tens of iterations (typically 30/40 iterations), with monotonically decreasing values of the learning factor α :

$$\alpha = 0.9 \cdot e^{-\frac{(i-1) \cdot \ln(9)}{(I-1)}}, \quad (10)$$

where i is the iteration number ($1 \leq i \leq I$). At each iteration, training was done using a batch approach (see e.g. [16], [17]), since sequential training would be computationally prohibitive for the millions of passwords in the training set.

In Figure 4, we can see the False Alarm probability (p_{FA}) given by a (140, 200) SOM, and different sized CM-sketches, when applied to the mentioned list of passwords, with classification thresholds corresponding to the popularity level of each one of the 100000 more popular passwords of the Rockyou set. For each threshold, all 32602348 passwords on the password list were classified as being dangerous (popularity above threshold) or not (popularity below threshold); the presented p_{FA} is the observed ratio of false positives. The x-axis in this figure corresponds to the ordinal position (in decreasing popularity level sort order) of the password whose popularity level is being used as threshold.

Two notes should be made, concerning this Figure. Firstly, we should consider the very different natures of a CM-sketch's false positive, and what we are calling a SOM's false positive. In the CM-sketch case, due to the uniformly distributed randomizing effect of the hash functions, false positives are purely stochastic events, occurring on uniform, flat spaces; in the SOM case, however, a false positive is not a purely stochastic event, since it typically will occur as a (non-stochastic) result of the closeness between the candidate password and an existing popular password. Hence, while a CM-Sketch false positive is a pure statistical classification error, a SOM false positive may not be an error at all, but a manifestation of the desirable generalisation properties that this construct possesses by design. Since any list with real world passwords will, in fact, possess many passwords bearing close proximity relations with popular passwords (e.g. the

popular choices "password" or "password1", and the much less popular "password8" or "password9"), a much bigger rate of false positives was expected from the SOM, when compared with the CM-sketch. However, as can be seen in Figure 4, the false alarm rate of the SOM is comparable with the false alarm rate of CM-sketches of an equivalent number of cells. In fact, the SOM even presents mostly lower false positives rates than, for example, the (3, 28000) CM-sketch, which has three times more cells than the SOM.

Secondly, we should point out that, since each output cell of a CM-Sketch is a single scalar, and each node in the SOM is a vector, the number of cells does not directly map the physical size of the compressed password list. For example, the (3, 56000) CM-sketch used in Figure 4 will typically require 672 kbytes, while the (140, 200) SOM, even though with only one sixth of the cells, did require 843.5 kbytes (each node requires 30 bytes for the model, plus one bit for its label).

The most noticeable feature to be appreciated in this comparison is, therefore, the considerably different behavior of the SOM and CM-sketch p_{FA} when the popularity threshold decreases, reflecting the different nature of the classification "errors" in each case. Insofar as compression rates go, both schemes can be considered roughly equivalent (even though with a clear advantage of the CM-sketch in the region of higher popularity thresholds)

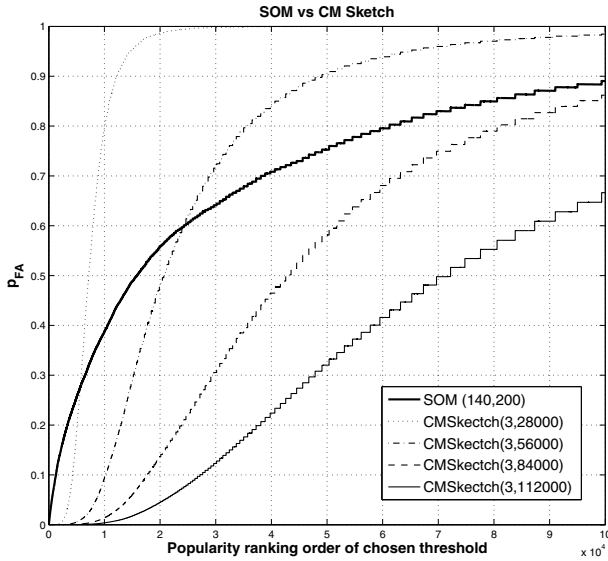


Fig. 4. Size requirements comparison

3.2 Generalisation Capability and Consistency

The fundamental difference between the proposed SOM based scheme and the previously existing CM-Sketch approach is the design objective of having the capability to

generalize, and flag as dangerous password choices similar to previously used popular ones. Hence, the evaluation of the achieved generalisation capability of the overall scheme is paramount.

Clearly, generalisation increases with increasing compression rate: smaller SOM will possess fewer nodes and, hence, each node will be chosen as the BMU for an increasing larger number of different entries in the training set. This means that they will become less and less specialized, and its model will, therefore, reflect an increasing larger neighborhood of passwords. On the other hand, a SOM with a large number of nodes, as compared to the size of the training set, will acquire a high degree of specialisation of its models. Each node will therefore be more narrowly tuned to smaller password neighborhoods. To increase specialisation (thus reducing generalisation), one must increase the size of the SOM; to increase generalisation (thus reducing specialisation), one must decrease the SOM size, or apply a smoothing window to the obtained SOM, such as the one presented in Section 2.2. This later alternative (the use of a smoothing window) tends to produce more consistent generalisation behaviour.

As is also clear, evaluating the consistency of the generalisation achieved by the SOM approach, implies answering the following two questions:

- Is the SOM generalising in a useful way? That is: is it flagging as dangerous, not only the previously popular passwords, but also the corresponding mutations of weak passwords that people are prone to use in their defacing attempts?
- Is the SOM generalising too much? That is: is it flagging as dangerous passwords which do not present any real danger and, thus, unduly limiting the space of available choices?

To answer to the first question, one needs to estimate the password mutations that people are prone to use, in their effort to deface their chosen passwords, without losing memorability. Such exercises have previously been done in several fora, though each choice of the set of mutation rules has a high level of subjectivity. A well known set of mutation rules is the one used by John-the-Ripper [18], a widely known and used password cracking software tool. To evaluate how the proposed scheme deals with the John the Ripper mutations, a (60,100) SOM was trained with the 32 million Rockyou passwords used earlier in this paper; the classification threshold was chosen as 1000 (the popularity level of the 1438th most popular password). The first 500 most popular passwords were then presented to John the Ripper, to produce all possible mutations. Lastly, these 500 passwords and all their mutations were classified by our SOM based scheme. Naturally, one would expect that i) all 500 passwords were flagged as dangerous (since $p_{miss} = 0$), and ii) most of their mutations are also flagged as dangerous (since the John-the-Ripper mutation rules are supposed to reasonably emulate user's password defacing choices). The obtained results can be seen in Table 1.

As can be seen in this table, not only were all previously known popular choices recognised as dangerous, but also the vast majority of their John-the-Ripper mutations were recognised as also being dangerous (not because they were historically popular, but simply by being *similar* to historically popular choices).

To address the second identified question, and determine if the SOM generalising too much, and, thus, unduly limiting the space of available choices, a set of 1 million

Table 1. Passwords flagged as dangerous

Passwords	Flagged as dangerous
500 most probable	100%
All mutations	84%

random passwords was generated. Each password was constituted by a string of random characters (ASCII codes comprised between 32 and 127), and with random lengths between 1 and 21: uniform distributions were used in both cases. Being random, the level of similarity with previously used popular passwords should be small and, hence, one would expect their vast majority to be approved as valid passwords. In fact, only 11.3% were considered dangerous by the (60,100) SOM trained with the Rockyou password list.

The previous results indicate, therefore, that the intended generalisation capability is operating properly: known popular passwords are forbidden; passwords which are similar to known popular passwords are mostly forbidden; random, non-structured passwords are mostly allowed.

As a last example of performance, and still using the same SOM, we classified wordlists of several languages, broadly covering the range from germanic to romanic Indo-European languages, with a total of 1.64 million words, from [19]. Since the SOM was trained with the Rockyou list, where english and spanish derived words are dominant, one would expect a higher percentage of forbidden passwords in the English and Spanish files, not only due to the eventual appearance, in those files, of passwords popular in the training set, but mainly due to syntactic similarities within the languages. The results can be seen in Figure 5. As expected, English and Spanish will provide the higher percentage of words considered dangerous, due to similarities with passwords on the training set. Also, we can see, from this picture, that the classifier seems to be more sensitive to words of romanic structure (right side of the picture), than to languages of germanic origin (left side of the picture). This tendency is not unexpected, since the training set is densely populated with words from english and spanish; what might be unexpected is the capability of the classifier to tune to the syntactic structure of the training set and, thus, display this type of selectivity. As a final note, we may note that, independently of the language, dictionary words tend to be more susceptible to be considered dangerous than random passwords, even if they are new to the classifier. This is, again, an expected result, since dictionary words will possess an underlying similar syntactic structure to dictionary words of similar languages appearing in the training set.

3.3 Classification Speed

Even though training of the classification database (SOM) is a lengthy, computationally heavy process, this step is centralised, and has no stringent timing requisites associated with it. The one step that must be light and fast is the final classification step, since users will locally execute it each time a password is tested. In the proposed scheme, however, the classification of a password as "allowable" or "not allowed" is extremely fast, and

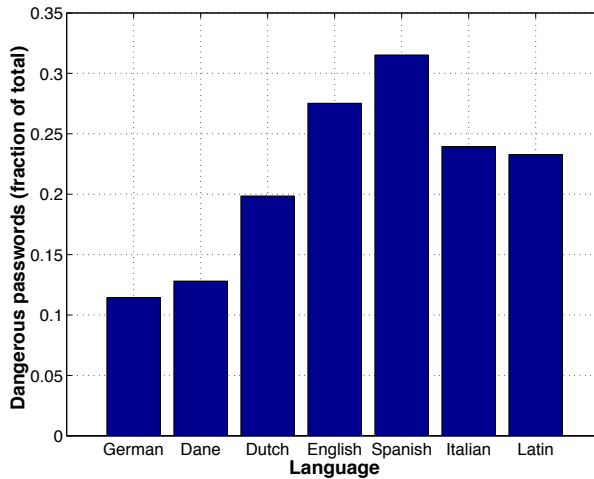


Fig. 5. Flagging in several languages

independent of the size of both the set of previously known passwords. The user sided operations are a simple DFT transform of a very short sequence (whose length is the size of the proposed password), and a search for the BMU in the classification database (whose size is in the order of a few thousand models). As such, classification speed is not an issue in this scheme, which can be used even by devices with very limited computing power or low latency requirements.

4 Proof of Concept Implementation

To empirically test the use of this password validation scheme, we created an independent implementation in the form of a Google Chrome browser extension. This implementation is available on the chrome webstore with the name: SOM Password Validator¹.

There were mainly three things that we wanted to show with this implementation: i) the scheme can be easily implemented; ii) local validation of passwords is fast; iii) and that the final size of the application database distributed to users is small.

The implementation is in javascript, and the final product has less than 500 lines of code. The time taken to validate each individual password is effectively negligible, taking less than one second to validate 1000 different passwords. The database shipped with the application has a total size of 1MB, bringing the total size of the application (with images included) to short of 2MB. A screenshot of the extension can be seen in Figure 6.

¹ The extension can be obtained here: <http://goo.gl/xjcGW8>

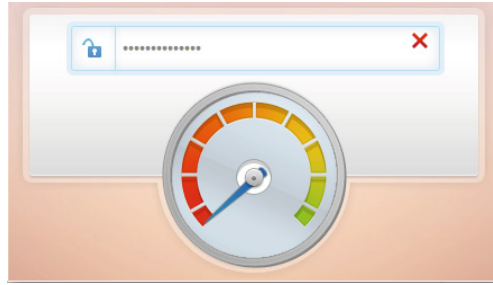


Fig. 6. Google Chrome browser extension implementation

5 Conclusions and Future Work

A scheme for password validation was presented. As in previous approaches to the problem, the proposed solution builds on the notion that frequent, common passwords constitute bad choices, and that rarely (or never) seen passwords should be favored, since they are more prone to resist to statistical dictionary attacks. However, our approach differs from existing proposals in that: i) it is envisaged for local, decentralized operation, and ii) it has generalisation capabilities. This generalisation capability not only allows detection of dangerous passwords by proximity to popular ones, but also avoids the need for frequent updates of the validation database, since it protects users from the evolutionary changes of the database between updates.

These goals, even though conceptually simple, imply the recourse to several concepts which are amenable to very different implementations and choices. The base tool used in this article (SOM) can be replaced by other clustering/mapping alternatives; the used similarity measure for non-categorical data is arbitrary in a large degree, and different alternatives could have been pursued; the advantages and implications of a preliminary feature-extraction step prior to SOM training can be discussed and argued; the hashing functions used in the masking step are also nothing else than arbitrary, even though effective; all in all, in almost all steps of the method, many and widely different options could have been made. As such, there is no claim of optimality in this article. In fact, if any claim is made, it is one of almost certainty of non-optimality, and of the need to further pursue potential gains in performance by further work concerning the choices to be made at each individual step. However, the approach has proved feasible. It is possible to build a password classification tool amenable to local operation, and capable of effectively facing the challenges imposed by the "mutation-capable" last generation of statistical dictionary based attacks.

References

1. Clair, L.S., Johansen, L., Enck, W., Pirretti, M., Traynor, P., McDaniel, P., Jaeger, T.: Password exhaustion: Predicting the end of password usefulness. In: Bagchi, A., Atluri, V. (eds.) ICISS 2006. LNCS, vol. 4332, pp. 37–55. Springer, Heidelberg (2006)

2. Castelluccia, C., Durmuth, M., Perito, D.: Adaptive password-strength meters from markov models. In: NDSS. The Internet Society (2012)
3. Spafford, E.H.: Opus: Preventing weak password choices. *Computers & Security* (1992)
4. Schechter, S., Herley, C., Mitzenmacher, M.: Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In: *Proceedings of the 5th USENIX Conference on Hot Topics in Security, HotSec 2010*. USENIX Association, Berkeley (2010)
5. Hashcat password recovery tool (2013), <http://hashcat.net/hashcat/>
6. Haykin, S.: *Neural Networks: A Comprehensive Foundation*, 2nd edn. Prentice Hall PTR, Upper Saddle River (1998)
7. Kohonen, T.: *Neurocomputing: Foundations of research*. MIT Press, Cambridge (1988)
8. Boriah, S., Chandola, V., Kumar, V.: Similarity measures for categorical data: A comparative evaluation. In: *Proceedings of the Eighth SIAM International Conference on Data Mining*
9. Huang, Z.: Extensions to the k-means algorithm for clustering large data sets with categorical values (1998)
10. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998*. ACM, New York (1998)
11. He, K., Wen, F., Sun, J.: K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In: *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Washington, DC (2013)
12. Boufounos, P., Rane, S.: Secure binary embeddings for privacy preserving nearest neighbors. In: *Proceedings of the 2011 IEEE International Workshop on Information Forensics and Security, WIFS 2011*. IEEE Computer Society, Washington, DC (2011)
13. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: *Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG 2004*. ACM, New York (2004)
14. Rockyou list of leaked passwords (2013), <https://wiki.skullsecurity.org/Passwords>
15. Cormode, G., Muthukrishnan, S.: An improved data stream summary: The count-min sketch and its applications. *J. Algorithms* (April 2005)
16. Kohonen, T.: Fast evolutionary learning with batch-type self-organizing maps. *Neural Process* (April 1999)
17. Fort, J.C., Letremy, P., Cottrell, M.: Advantages and drawbacks of the batch kohonen algorithm. In: Verleysen, M. (ed.) *ESANN* (2002)
18. John the ripper password cracking tool, <http://www.openwall.com/john/>
19. Openwall wordlist collection, <http://www.openwall.com/wordlists/>