Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries

Dario Pasquini, *Sapienza University of Rome, Institute of Applied Computing CNR;* Marco Cianfriglia, *Institute of Applied Computing CNR;* Giuseppe Ateniese, *Stevens Institute of Technology;* Massimo Bernaschi, *Institute of Applied Computing CNR*

https://www.usenix.org/conference/usenixsecurity21/presentation/pasquini

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries

Dario Pasquini[†,§], Marco Cianfriglia[§], Giuseppe Ateniese[‡] *and* Massimo Bernaschi[§]
[†]*Sapienza University of Rome,* [‡]*Stevens Institute of Technology,* [§]*Institute of Applied Computing CNR*

Abstract

Password security hinges on an in-depth understanding of the techniques adopted by attackers. Unfortunately, real-world adversaries resort to pragmatic guessing strategies such as dictionary attacks that are inherently difficult to model in password security studies. In order to be representative of the actual threat, dictionary attacks must be thoughtfully configured and tuned. However, this process requires a domain-knowledge and expertise that cannot be easily replicated. The consequence of inaccurately calibrating dictionary attacks is the unreliability of password security analyses, impaired by a severe measurement bias.

In the present work, we introduce a new generation of dictionary attacks that is consistently more resilient to inadequate configurations. Requiring no supervision or domain-knowledge, this technique automatically approximates the advanced guessing strategies adopted by real-world attackers. To achieve this: (1) We use deep neural networks to model the proficiency of adversaries in building attack configurations. (2) Then, we introduce dynamic guessing strategies within dictionary attacks. These mimic experts' ability to adapt their guessing strategies on the fly by incorporating knowledge on their targets.

Our techniques enable more robust and sound password strength estimates within dictionary attacks, eventually reducing overestimation in modeling real-world threats in password security.

1 Introduction

Passwords have proven to be irreplaceable. They are still preferred over safer options and appear essential in fallback mechanisms. However, users tend to select their passwords as easy-to-remember strings, which results in very skewed distributions that an attacker can easily model. This makes passwords and authentication systems that implement them inherently susceptible to guessing attacks. In this scenario, the security of the authentication protocol cannot be stated via a security parameter (e.g., the key size). The only way to establish the soundness of a system is to model adversarial behaviors and cast accurate adversary models. To this end, simulating password guessing attacks has become a pivotal task.

In this direction, more than three decades of active research provided us with powerful password models [28, 31, 32, 44]. However, very little progress has been made to systematically model real-world attackers and their guessing strategies [26, 41]. As a matter of fact, *password crackers* rarely harness fully-automated approaches developed in academia. They rely on more pragmatic guessing techniques that present stronger inductive biases. In offline attacks, experts use high-throughput, and extremely flexible techniques such as **dictionary attacks with mangling rules** [29]. This class of attacks produces candidate passwords by expanding a dictionary/wordlist through a set of scripted string transformations (a rules-set) which aim at mimicking users' composition habits such as *leeting* (e.g., "*pa$$w0rd*") or concatenating digits (e.g., "*password123*") [17].

Unlike fully-automated approaches, dictionary attacks are heavily sensitive to their initial configuration. To be effective, these must rely on highly tuned setups—pairs of dictionaries and mangling rules-sets that have been carefully optimized and thoroughly calibrated. To cast such configurations, real-world attackers rely on a manual process that is based on specific expertise that can only be achieved and refined over years of practical experience [3]. Furthermore, attackers customize their configurations for the current target by dynamically adjusting the dictionary and rules-set leveraging information gathered before or during the attack.

Unfortunately, lacking the same domain-knowledge of experts, most researchers and security practitioners performing dictionary attacks in their security analysis rely on *off-the-shelf* setups and static guessing strategies that only remotely approximate the actual effectiveness of real-world attacks. Indeed, as demonstrated in [41], these commonly used default configurations bring to a profound overestimation of password strength that fails to correctly

approximate adversarial capabilities. Unavoidably, this introduces a strong **bias** in the produced strength estimates that fundamentally sways the conclusion of security analysis.

In the present paper, we move towards reducing this inherent measurement bias by devising a new generation of dictionary attacks that automates the advanced guessing strategies adopted by attackers; we cast an adversary model that is consistently more resilient to inaccurate configurations, and that better describes real-world attackers' capabilities. To that purpose, we introduce general procedures that systematically mimic different adversarial behaviors:

First, by relying on deep learning techniques, we devise the **Adaptive Mangling Rules attack**. This artificially simulates the optimal configurations harnessed by expert adversaries by explicitly handling **the conditional nature of mangling rules**. Here, during the attack, each word from the dictionary is associated with a dedicated and possibly unique rules-set created at runtime via a **deep neural network.** Using this technique, we confirmed that standard attacks, based on *off-the-shelf* dictionaries and rules-sets, are sub-optimal and can be easily compressed up to an order of magnitude in the number of guesses. Furthermore, we are the first to explicitly model the strong relationship that binds mangling rules and dictionary words, demonstrating its connection with optimal configurations in dictionary attacks.

Then, we introduce **dynamic guessing strategies** within dictionary attacks [32]. Real-world adversaries perform their guessing attacks incorporating prior knowledge on the targets and dynamically adjusting their guesses during the attack. In doing so, professionals seek to optimize their configurations and maximize the number of compromised passwords. Unfortunately, automatic guessing techniques fail to model this adversarial behavior. Instead, we demonstrate that dynamic guessing strategies can be enabled in dictionary attacks and substantially improve the guessing attack's effectiveness even without prior optimization. More prominently, our technique makes dictionary attacks consistently more resilient to misconfigurations by promoting the completeness of the dictionary at runtime.

Finally, we combine these general methodologies and introduce the Adaptive Dynamic Mangling rules attack (*AdaMs*). The *AdaMs* attack consistently reduces the overestimation induced by sub-optimal configurations in dictionary attacks, enabling more reliable and sound password strength estimates.

Organization: Section 2 gives an overview of the fundamental concepts needed for the comprehension of our contributions. In Section 3, we introduce Adaptive Mangling Rules aside the intuitions and tools on which those are based. Section 4 discusses dynamic mangling rules attacks. Finally, Section 5 aggregates the previous methodologies, introducing the *AdaMs* attack. The motivation and evaluation of the proposed techniques are presented in their respective sections. Section 6 concludes the paper, although supplementary information is provided in the Appendices.

2 Background and preliminaries

We start by covering password guessing attacks and their foundations in Section 2.1. In Section 2.2, we focus on dictionary attacks that are the basis of our contributions. Next, Section 2.3 briefly discusses relevant related works. Finally, we define the threat model in Section 2.4.

2.1 Password Guessing

Human-chosen passwords do not distribute uniformly in the exponentially large key-space. Users tend to choose easy-to-remember passwords that aggregate in relatively few dense clusters. Real-world passwords, therefore, tend to cluster in very bounded distributions that can be modeled by an attacker, making authentication-systems intrinsically susceptible to **guessing attacks**. In a guessing attack, the attacker aims at recovering plaintext credentials by attempting several candidate passwords (guesses) till success or *budget* exhaustion; this happens by either searching for collisions of password hashes (**offline attack**) or attempting remote logins (**online attack**). In this process, the attacker relies on a so-called **password model** that defines which, and in which order, guesses should be tried to maximize the effectiveness of the attack (see Section 2.4).

Generally speaking, a password model can be understood as a suitable estimation of the password distribution that enables an educated exploration of the key-space. Existing password models construct over a heterogeneous set of assumptions and rely on either intuitive or rigorous security definitions. From the most practical point of view, those can be divided into two macro-classes: parametric and nonparametric password models.

Parametric approaches build on top of probabilistic reasoning; they assume that real-world password distributions are sufficiently smooth to be accurately described from suitable parametric probabilistic models. Here, a password mass function is explicitly [28, 31] or implicitly [32] derived from a set of observable data (i.e., previously leaked passwords) and used to assign a probability to each element of the key-space. During the guessing attack, guesses are produced by traversing the key-space following the decreasing probability order imposed by the modeled mass function. These approaches are, in general, relatively slow and unsuitable for practical offline attacks. Although simple models such as Markov Chains can be employed [19], more advanced and effective models such as the neural network ones [28,32] are hardly considered outside the research domain due to their inefficiency.

Nonparametric models such as Probabilistic Context-Free Grammars (PCFG) and dictionary attacks rely on simpler and

Rule	Result	Rule description.
`r`	"*niemtel*"	Reverse string.
`T0`	"*Letmein*"	Capitalize the first character.
`$9 $9`	"*letmein99*"	Append "*99*" to the string.
`se3`	"*l3tm3in*"	Substitute the character *'e'* with *'3'*.
`] ] ] $m $a $n`	"*letmman*"	Remove the last three symbols and append the string "*man*".

Table 1: Example of mangling rules and their effect on the dictionary-word "*letmein*". The rules are selected from the rules-set *Best64*.

more intuitive constructions, which tend to be closer to human logic. Generally, those assume passwords as realizations of templates and generate novel guesses by abstracting and applying such patterns on ground-truth. These approaches maintain a collection of tokens that are either directly given as part of the model configuration (e.g., the dictionary and rules-set for dictionary attack.) or extracted from observed passwords in a setup phase (e.g., terminals/grammar for PCFG). In contrast with parametric models, these can produce only a limited number of guesses, which is a function of the chosen configuration. A detailed discussion on dictionary attacks follows in the next section.

2.2 Dictionary Attacks

Dictionary attacks can be traced back to the inception of password security studies [29, 39]. They stem from the observation that users tend to pick their passwords from a bounded and predictable pool of candidates; common natural words and numeric patterns dominate most of this skewed distribution [38]. An attacker, collecting such strings (i.e., creating a dictionary/wordlist), can use them as high-quality guesses during a guessing attack, rapidly covering the key-space's densest zone. These dictionaries are typically constructed by aggregating passwords revealed in previous incidents and plain-word dictionaries.

Although dictionary attacks can produce only a limited number of guesses[1], these can be extended through **mangling rules**. Mangling rules attacks describe password distributions by factorizing guesses in two main components: (1) dictionary-words and (2) string transformations (mangling rules). These transformations aim at replicating users' composition behaviors. Mangling transformations are modeled by the attacker and collected in sets (rules-sets). During the guessing attack, each dictionary word is extended in real-time through mangling rules, creating novel guesses that augment the guessing attack's coverage over the key-space. Hereafter, we use the terms dictionary attack and mangling rules attack interchangeably.

[1]The required disk space inherently bounds the number of guesses issued from plain dictionary attacks. Guessing attacks can quickly go beyond 10^{12} guesses, and storing such a quantity of strings is not practical.

Most widely known implementations of mangling rules are included in the *password cracking* software *Hashcat* [15] and *John the Ripper* [18] (JtR). Here, mangling rules are encoded through simple custom programming languages. Table 1 reports some instances of mangling rules and their effect. *Hashcat* and *JtR* share almost overlapping mangling rules languages, although few peculiar instructions are unique to each tool. However, they consistently differ in the way mangling rules are applied during the attack. *Hashcat* follows a **word-major order**, where all the rule-set rules are applied to a single dictionary-word before the next dictionary word is considered. In contrast, *JtR* follows a **rule-major order**, where a rule is applied to all the dictionary words before moving to the next rule. In our work, we rely on the approach of *Hashcat* as the word-major order is necessary to efficiently implement the adaptive mangling rules attack that we introduce in Section 3.3.

The community behind these software packages developed numerous mangling rules sets that have been made public. Such sets have a heterogeneous size and can range between tens to thousands of entries. Mangling rules can be either manually crafted by human experts and optimized through public competitions [8] or produced via simple automatic procedures [4]. Here, it is important to note that public rules-sets are often sub-optimal when compared to highly-tuned, private sets harnessed by experts [26].

Despite their simplicity, mangling rules attacks represent a substantial threat in offline password guessing. Mangling rules are swift and inherently parallel; they are naturally suited for both parallel hardware (i.e., GPUs) and distributed setups, making them one of the few guessing approaches suitable for large-scale attacks (e.g., botnets).

Furthermore, real-world attackers update their guessing strategy dynamically during the attack [41]. Basing on prior knowledge and the initially matched passwords, they tune their guesses generation process to describe their target set of passwords better and eventually recover more of them. To this end, professionals prefer extremely flexible tools that allow for fast and complete customization. While the state-of-the-art probabilistic models fail at that, dictionary attacks make any form of customization feasible as well as natural.

2.3 Related Works

Although dictionary attacks are ubiquitous in password security research [9, 12, 14, 23, 28], little effort has been spent studying them. This section covers the most relevant contributions.

Ur et al. [41] firstly made explicit the large performance gap between optimized and stock configurations for mangling rules attacks. In their work, Ur et al. recruited professional figures in password recovery and compared their performance against *off-the-shelf* parametric/nonparametric approaches in different guessing scenarios. Here, professional attackers have

Name	Unique Passwords	Brief Description
LinkedIn [25]	60.599.259	An employment-oriented online service.
youku [48]	47.487.499	Chinese video hosting service.
MyHeritage [30]	36.393.972	Online genealogy platform.
zooks [50]	29.010.979	Online dating service available.
RockYou [36]	14.344.391	Gaming platform.
animoto [2]	8.420.466	A cloud-based video creation service.
zomato [49]	4.955.821	Indian, food delivery application. About 40% of the password are random tokens of six alphanumeric characters.
phpBB	184.389	Software website.

Table 2: Password leaks used in the paper sorted by size.

been shown capable of vastly outperform any password model. This thanks to custom dictionaries, proprietary mangling rules, and the ability to create tailored rules for the attacked set of passwords. Finally, the authors show that the performance gap between professional and non-professional attackers can be reduced by combining guesses of multiple password models.

More recently, Liu et al. [26] produced a set of tools that can be used to optimize the configuration of dictionaries attacks. These solutions extend previous approaches [4, 37], making them faster. Their core contribution is an algorithm capable of inverting almost all mangling rules; that is, given a rule r and password to evaluate p, the inversion-rule function produces as output a *regex* that matches all the preimages of $r(p)$ i.e., all the dictionary entries that transformed by r would produce p. At the cost of an initial pre-computation phase, following this approach, it is possible to count dictionary-words/mangling-rules *hits* (i.e., guessed passwords) on an attacked set without enumerating all the possible guesses. Liu et al. used the method to optimize the ordering of mangling rules in a rules-set by sorting them in decreasing hits-count order.[2] In doing so, the authors observed that default rules-sets follow an optimal ordering only rarely.

Basing on the same general approach, they speedup the automatic generation of mangling rules [4] and augment dictionaries by adding missing words in consideration of known attacked sets [37]. Similarly, they derive an approximate guess-number calculator for rule-major order attacks.

2.4 Threat Model

In our study, we primarily model the case of trawling, offline attacks. Here, an adversary aims at recovering a set of passwords $\mathbf{X}$ (also referred to as *attacked-set*) coming from an arbitrary password distribution $P(\mathbf{x})$ by performing a guessing attack. To better describe both the current trend in password storing techniques [20, 34, 35] and real-world attackers' goals [5], we assume a rational attacker who is bound to produce a limited number of guesses. More precisely, this attacker aims at maximizing the number of guessed passwords in $\mathbf{X}$ given a predefined *budget* i.e., a maximal number of guesses the attacker is willing to perform on $\mathbf{X}$. Hereafter, we model this strategy under the form of β-*success-rate* [6, 7]:

$$s_\beta(X) = \sum_{i=1}^{\beta} P(x_i). \quad (1)$$

Experimental setup In our construction, we do not impose any limitation on the nature of $P(\mathbf{x})$ nor the attacker's *a priori* knowledge. However, in our experiments, we consider a weak attacker who does not retain any initial knowledge of the target distribution i.e., who cannot provide an optimal attack configuration for $\mathbf{X}$ before the attack. This last assumption makes a better description of the use-case of automatic guessing approaches currently used in password security studies.

In the attacks reported in the paper, we always sort the words in the dictionary according to their frequency. Additionally, in the reported results for all the dictionary attacks, we do not count guesses that remain unchanged after the application of a mangling rule ($r(w) = w$). This aims to avoid biases in measuring the effectiveness of the adaptive approach presented in Section 3.3. The password leaks that we use through the paper are listed in Table 2.

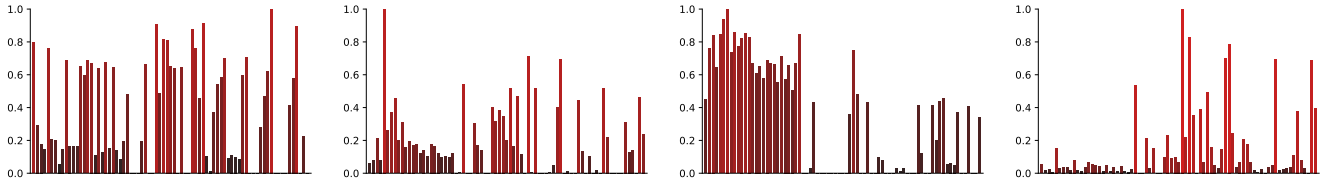
3 The Adaptive Mangling Rules attack

This section introduces the first core block of our password model: the Adaptive Mangling Rules. We start in Section 3.1, where we make explicit the conditional nature of mangling rules while discussing its connection with optimal attack configurations. In Section 3.2, we model the functional relationship connecting mangling rules and dictionary words via a deep neural network. Finally, leveraging the introduced tools, we establish the Adaptive Mangling Rules attack in Section 3.3.

Motivation: Dictionary attacks are highly sensitive to their configuration; while parametric approaches tend to be more robust to training sets and hyper-parameters choices, the performance of dictionary attacks crucially depends on the selected dictionary and rules-set [26, 41]. As evidenced by Ur et al. [41], real-world attackers rely on extremely optimized configurations. Here, dictionaries and mangling rules are jointly created over time through practical experience [3], harnessing a domain knowledge and expertise that is mostly unknown to the academic community [26].

Password security studies often rely on publicly available dictionaries and rules-sets that are not as effective as advanced configurations adopted by professionals. Unavoidably, this leads to a constant overestimation of password strength that skews studies and reactive analysis conclusions.

[2] Primarily, for rule-major order setups (e.g., JtR).



(a) Only digits. (b) Only capital letters. (c) Strings of length 5. (d) Strings of length 10.

Figure 1: Distribution of hits per rule for 4 different input dictionaries for the same attacked-set i.e., *animoto*. Within a plot, each bar depicts the normalized number of hits for one of the 77 mangling rules in *best64*. We performed the attack with *Hashcat*.

Hereafter, we show that professional attackers' domain-knowledge can be suitably approximated with a **Deep Neural Network**. Given that, we devise a new dictionary attack that autonomously promotes functional interaction between the dictionary and the rules-set, implicitly simulating the precision of real-world attackers' configurations.
We start by presenting the intuition behind our technique. Formalization and methodology are reported later.

3.1 The conditional nature of mangling rules

As introduced in Section 2.2, dictionary attacks describe password distributions by factorizing guesses into two main components—a dictionary word w and a transformation rule r. Here, the word w acts as a **semantic base**, whereas r is a **syntactic transformation** that aims at providing a suitable guess through the manipulation of w. Generally speaking, such factorized representation can be thought of as an approximation of the typical users' composition behavior: starting from a plain word or phrase, users manipulate it by performing operations such as *leeting*, appending characters or concatenation.

At configuration time, such transformations are abstracted and collected in arbitrary large rules-sets under the form of mangling rules. Then, during the attack, guesses are reproduced by exhaustively applying the collected rules to all the dictionary words. **In this generation process, rules are applied unconditionally on all the words, assuming that the abstracted syntactic transformations equally interact with all the dictionary elements.**

However, arguably, users do not follow the same simplistic model in their password composition process. Users first select words and then mangling transformations conditioned by those words. That is, mangling transformations are subjective and depend on the base words on which those are applied. For instance, users may prefer to append digits at the end of a name (e.g., "*jimmy*" to "*jimmy91*"), repeat short words rather than long ones (e.g., "*why*" to "*whywhywhy*") or capitalize certain strings over others (e.g., "*cookie*" to "*COOKIE*"). A similar intuition was harnessed in [43], where the semantic of words was considered in defining context-free grammars for passwords.

In this direction, we can think of each mangling rule as a function that is valid on an arbitrary small subset of the dictionary space, strictly defined by the users' composition habits. Thus, applying a mangling rule on words outside this domain unavoidably brings it to produce guesses that have only a negligible probability of inducing hits during the guessing attack (i.e., that do not replicate users' behavior). This concept is captured in Figure 1, where four panels depict the *hits* distribution of the rules-set "*best64*" for four different dictionaries. Each dictionary represents a specific subset of the dictionary space that has been obtained by filtering out suitable strings from the *RockYou* leak; namely, these are passwords composed of: digits (Figure 1a), capital letters (Figure 1b), passwords of length 5 (Figure 1c), and passwords of length 10 (Figure 1d). The four histograms show how mangling rules selectively and heterogeneously interact with the underlying dictionaries. Rules that produce many hits for a specific dictionary inevitably perform very poorly with the others.

Eventually, the conditional nature of mangling rules has a critical impact in defining the effectiveness of a dictionary attack. To reach optimal performance, an attacker has to resort to a setup that *a priori* maximizes the conditional effectiveness of mangling rules. In this direction, we can see highly optimized configurations used by experts as pairs of dictionaries and rules-sets that organically support each other in the guesses generation process.[3] On the other hand, configurations based on arbitrary chosen rule-sets and dictionaries may not be fully compatible, and, as we show later in the paper, they generate many low-quality guesses. Unavoidably, this phenomenon makes adversary models based on mangling rules inaccurate and induce an overestimation of password strength [41].

Next, we show how modeling the conditional nature of mangling rules allows us to cast dictionary attacks that are inherently more resilient to poor configurations.

[3] This has also been indirectly observed by Ur et al. in their ablation study on pro's guessing strategy, where the most remarkable improvement was achieved with a proprietary dictionary in tandem with a proprietary rules-set.

3.2 A Model of Rule/Word Compatibility

We introduce the notion of **compatibility** that refers to the functional relation among dictionary words and mangling rules discussed in the previous section. The compatibility can be thought of as a continuous value defined between a mangling rule r and a dictionary-word w that, intuitively, measures the *utility* of applying the rule r on w. More formally, we model *compatibility* as a function:

$$\pi : R \times \mathbb{W} \rightarrow [0,1],$$

where R and $\mathbb{W}$ are the rule-space (i.e., the set of all the suitable transformations $r : \mathbb{W} \rightarrow \mathbb{W}$) and the dictionary-space (i.e., the set of all possible dictionary words), respectively. Values of $\pi(w,r)$ close to 1 indicate that the transformation induced by r is well-defined on w and would lead to a valuable guess. Values close to 0, instead, indicate that users would not apply r over w, i.e., guesses will likely fall outside the dense zone of the password distribution.

This formalization of the compatibility function also leads to a straightforward probabilistic interpretation that better supports the learning process through a neural network. Indeed, we can think of π as a probability function over the event:

$$r(w) \in \mathbf{X},$$

where $\mathbf{X}$ is an attacked set of passwords. More precisely, we have that:

$$\forall_{w \in \mathbb{W},\ r \in R} \big(\pi(r,w) = P(r(w) \in \mathbf{X})\big).$$

In other words, $P(r(w) \in \mathbf{X})$ is the probability of guessing an element of $\mathbf{X}$ by trying the guess $g = r(w)$ produced by the application of r over w. Furthermore, such a probability can be seen as an unnormalized version of the password distribution, creating a direct link to probabilistic password models [28,31] as we have that:

$$\forall_{w \in \mathbb{W},\ r \in R} \langle \frac{\pi(r,w)}{Z} = P(r(w)) \rangle$$

for an intractable partition function Z. This follows from the observation that:

$$\forall g_i, g_j \in \mathbb{X} : P(g_i) \geq P(g_j) \Leftrightarrow P(r_i(x_i) \in \mathbf{X}) \geq P(r_j(x_j) \in \mathbf{X})$$
$$\text{with :}\ \ g_i = r_i(x_i) \text{ and } g_j = r_j(x_j),$$

where $\mathbb{X}$ is the key-space. However, this password distribution is defined over the factorized domain $R \times \mathbb{W}$ rather than directly over the key-space. This factorized form offers us practical advantages over the classic formulation. More in detail, by choosing and fixing a specific rule-space R (i.e., a rules-set), we can reshape the compatibility function as:

$$\pi_R : \mathbb{W} \rightarrow [0,1]^{|R|}. \tag{2}$$

This version of the compatibility function takes as input a dictionary-word and outputs a compatibility value for each rule in the chosen rule-set with **a single inference**. This form is concretely more computational convenient and will be used to model the neural approximation of the compatibility function.

Next, we show how the compatibility function can be inferred from raw data using deep learning.

3.2.1 Learning the compatibility function

As stated before, the probabilistic interpretation of the compatibility function makes it possible to learn π using a neural network. Indeed, the probability $P(r(w) \in \mathbf{X})$, in any form, can be described through a binary classification. That is, for each pair word/rule (w, r), we have to predict one of two possible outcomes: $g \in \mathbf{X}$ or $g \notin \mathbf{X}$, where $g = r(w)$. In solving this classification task, we can train a neural network in a logistic regression and obtain a good approximation of the probability $P(r(w) \in \mathbf{X})$.

In the same way, the reshaped formulation of π (i.e., Eq. 2) describes a **multi-label classification**. In multi-label classification, each input participates simultaneously to multiple binary classifications; an input is associated with multiple classes at the same time. More formally, having a fixed number of possible classes n, each data point is mapped to a binary vector in $\{0,1\}^n$. In our case, $n = |R|$ and each bit in the binary vector corresponds to the outcome of the event $r_j(w) \in X$ for a rule $r_j \in R$.

To train a model, then, we have to resort to a supervised learning approach. We have to create a suitable training-set composed of pairs *(input,label)* that the neural network can model during the training. Under our construction, we can easily produce such suitable labels by performing a mangling rules attack. In particular, fixed a rules-set R, we collect pairs (w_i, y_i), where w_i is the input to our model (i.e., a dictionary-word) and y_i is the label vector associated with w_i. As explicated before, the label y_i asserts the membership of the list of guesses $[r_1(w_i), r_2(w_i), \ldots, r_{|R|}(w_i)]$ over a hypothetical target set of passwords $\mathbf{X}$:

$$y_i = [r_1(w_i) \in \mathbf{X},\ r_2(w_i) \in \mathbf{X},\ \ldots,\ r_{|R|}(w_i) \in \mathbf{X}] \tag{3}$$

To collect labels, we have to concertize $\mathbf{X}$ by choosing a representative set of passwords. Intuitively, such a set should be as large and diverse as possible as it aims at describing the entire key-space. Hereafter, we refer to this set as X_A. This is the set of passwords we attack during the process of collecting labels. Similarly, we have to choose another set of strings W that represents the dictionary-space. This is used as input to the neural network during the training process and as the input dictionary during the simulated guessing attack. Details on the adopted set are given at the end of the section.

Finally, given X_A and W, and chosen a rules-space R, we construct the set of labels by simulating a guessing attack; that is, for each entry w_i in the dictionary W, we collect the label vector y_i (E.q. 3). In doing so, we used a modified version

Name	Cardinality	Brief Description
PasswordPro	3120	Manually produced.
generated	14728	Automatically generated.
generated2	65117	Automatically generated.

Table 3: Used *Hashcat*'s mangling rules sets.

of *Hashcat* described in Appendix E. Alternatively, the technique proposed in [26] can be used to speed up the collection of the labels.

Unlike the actual guessing attack, in the process, we do not remove passwords from X_A when those are guessed correctly; that is, the same password can be guessed multiple times by different combinations of rules and words. This is necessary to correctly model the functional compatibility. In the same way, we do not consider the identity mangling rule (i.e., *':'*) in the construction of the training set. When it occurs, we remove it from the rules set. To the same end, we do not consider hits caused by *conditional identity transformations* i.e., $r(w) = w$.

Training set configuration The creation of a training set entails the proper selection of the sets X_A and W as well as the rules-set R. Arguably, the most critical choice is the set X_A, as this is the ground-truth on which we base the approximation of the compatibility function. In our study, we select X_A to be the password leak discovered by *4iQ* in the *Dark Web* [1]. We completely anonymized all entries by removing users' information and obtained a set of $\sim 4 \cdot 10^8$ of *unique* passwords. We use this set as X_A within our models.
Similarly, we want W to be a good description of the dictionary-space. However, in this case, we are supported by the generalization capability of the neural network that can automatically obtain a more general description of the input space. In our experiments, we use the *LinkedIn* leak as W.

Finally, we train three neural networks that learn the compatibility function for three different rules-sets; namely *PasswordPro*, *generated* and *generated2*. Those sets are provided with the *Hashcat* software and widely studied in previous works [26, 28, 32]. Table 3 lists them along with some additional information.

Eventually, the labels we collect in the guessing process are extremely sparse. In our experiments, more than 95% of the guesses are a miss, causing our training-set to be extremely unbalanced towards the negative class.

Model definition and training We construct our model over a residual structure [16] primarily composed of mono-dimensional convolution layers. Here, input strings are first embedded at character-level via an embedding matrix; then, a series of residual blocks are sequentially applied to extract a global representation for dictionary words. Finally, such representations are mapped into the label-space by means of a single, linear layer that performs the classification task. To note that, although the model applies over sequential data, the use of a convolutional network instead of a recurrent one is essential to reduce inference latency. This will be critical in the context of our application (see Section 3.3).

This architecture is trained in a multi-label classification; each output of the final dense layer is squashed in the interval $[0,1]$ via the sigmoid function, and binary cross entropy is applied to each probability separately. The network's loss is then obtained by summing up all the cross-entropies of the $|R|$ classes/rules.

As mentioned in the previous section, our training-set is extremely unbalanced toward the negative class; that is, the vast majority of the ground-truth labels assigned to a training instance are negative (i.e., the application of a rule on the word does not bring to a hit). Additionally, a similar disproportion appears in the distribution per rule. Typically, we have many rules that count only a few positive examples, whereas others have orders of magnitude more hits. In our framework, we alleviate the negative effects of those disproportions by inductive bias. In particular, we achieve it by considering a **focal regulation** in our loss function [24].

Originally developed for object detection tasks in which there is a strong imbalance between foreground and background classes, we adopt focal regulation to account for sparse and underrepresented labels when learning the compatibility function. This *focal loss* is mainly characterized by a modulating factor γ that dynamically reduces the importance of well-classified instances in the computation of the loss function, allowing the model to focus on hard examples (e.g., underrepresented rules). More formally, the form of regularized binary cross entropy that we adopt is defined as:

$$\mathrm{FL}(p_j, y_j) = \begin{cases} -(1-\alpha)(1-p_j)^{\gamma}\log(p_j) & \text{if } y_j = 1 \\ \alpha p_j^{\gamma}\log(1-p_j) & \text{if } y_j = 0 \end{cases},$$

where p_j is the probability assigned by the model to the j'th class, and y_j is the ground-truth label (i.e., 1/hit and 0/miss). The parameter α in the equation allows us to declare an *a priori* importance factor to the negative class. We use that to down-weighting the correct predictions of the negative class in the loss function that would be dominant otherwise. In our setup, we dynamically select α based on the distribution of the hits observed in the training set. In particular, we choose $\alpha = \frac{\bar{p}}{(1-\bar{p})}$, where $\bar{p}$ is the ratio of positive labels (i.e., hits/guesses) in the dataset. Differently, we fix $\gamma = 2$ as we found this value to perform well via empirical evaluation.

Summing up, our loss function is defined as:

$$\mathcal{L}_f = \mathbb{E}_{x,y} \sum_{j=1}^{|R|} \mathrm{FL}(sigmoid(f(x)_j), y_j)$$

where f are the *logits* of the neural network. We train the model using *Adam* stochastic gradient descent [22] until an

early-stopping-criteria based on the *AUC* computed on a validation set is reached.

Maintaining the same general architecture, we train different networks with different sizes. In our experiments, we noticed that large networks provide a better approximation of the compatibility function, although small networks can be used to reduce the computational cost with a limited loss in utility. This suggests that modeling compatibility between rules and words is complex and that simpler models with less capacity (e.g., not based on deep neural networks) should perform poorly. In the paper, we report the results only for our biggest networks.

We implemented our framework on *TensorFlow*; the models have been trained on a *NVIDIA DGX-2* machine. A complete description of the architectures employed is given in Appendix B.

Ultimately, we obtain three different neural networks: one for each rule-set reported in Table 3. The suitability of these neural approximations will be proven later in the paper.

Additional approaches To improve the performance of our method, we further investigated domain-specific constructions for multi-label classification. In particular, we tested label embedding techniques together we deep architectures. Those are approaches that aim at modeling, implicitly, the correlation among labels. However, although unconditional dependence is evident in the modeled domain, we found no concrete advantage in considering it during the training. In the same direction, we investigated more sophisticated embedding techniques, where labels and dictionary-words were jointly mapped to the same latent space [47], yet achieving similar or worse performance.

Additionally, we tested implementations based on *transformer* networks [42], obtaining no substantial improvement. We attribute such a result to the lack of dominant long-term relationships among characters composing dictionary-words. In such a domain, we believe convolutional filters to be fully capable of capturing characters' interactions. Furthermore, convolutional layers are significantly more efficient than the multi-head attention mechanism used by *transformer* networks.

3.3 Adaptive Mangling Rules

As motivated in Section 3.2, each word in the dictionary interacts just with a limited number of mangling transformations that are conditionally defined by users' composition habits. While modern rules-sets can contain more than ten thousand entries, each dictionary-word w will interact only with a small subset of **compatible rules**, say R_w. As stated before, optimized configurations compose over pairs of dictionaries and rule-sets that have been created to mutually support each other. This is achieved by implicitly maximizing the average cardinality of the compatible set of rules R_w for each dictionary-word w in the dictionary.

In doing so, advanced attackers rely on domain knowledge and intuition to create optimized configurations. But, thanks to the explicit form of the compatibility function, it is possible to simulate their expertise. The intuition is that, given a dictionary-word w, we can infer the **compatible rules-set** R_w (i.e., the set of rules that interact well with w) according to the compatibility scores assigned by the neural approximation of π. More formally, given π for the rules-set R and a dictionary-word w, we can determine the compatible rules-set for w by *thresholding* the compatibility values assigned by the neural network to the rules in R:

$$R_w \approx R_w^\beta = \{r \mid r \in R \wedge \pi(w,r) > (1-\beta)\}, \qquad (4)$$

where $\beta \in (0,1]$ is a threshold parameter whose effect will be discussed later.

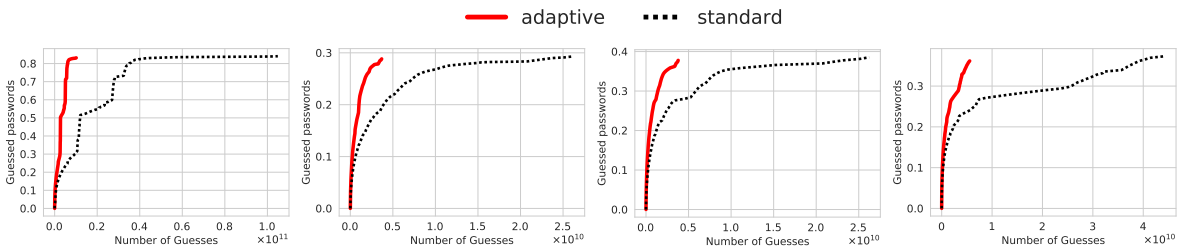
At this point, we simulate high-quality configuration attacks by ensuring dictionary-words does not interact with rules outside its compatible rules-set R_w^β. Algorithm 1 implements this strategy by following a word-major order in the generation of guesses. Every dictionary-word is limited to interact with the subset of compatible rules R_w^β that is decided by the neural net. **Intuitively, this is equivalent to assigning and applying a dedicated (and possibly unique) rules-set to each word in the dictionary.** Note that, the selection of the compatible rules-set is performed at runtime, during the attack, and does not require any pre-computation. We call this novel guessing strategy **Adaptive Mangling Rules**, since the rule-set is continuously adapted during the attack to better assist the selected dictionary.

The efficacy of *adaptive* mangling rules over the standard attack is shown in Figure 2, where multiple examples are reported. The adaptive mangling rules reduce the number of produced guesses while maintaining the hits count mostly unchanged. In our experiments, the adaptive approach induces compatible rules-sets that, on average, are an order of magnitude smaller than the complete rules-set. Typically, for β=0.5, only $\sim 10\%/15\%$ of the rules are conditionally applied to the dictionary-words. Considering the percentage of guessed passwords for adaptive and non-adaptive attacks, this means that approximately 90% of guesses are wasted during classic, unoptimized mangling rules attacks. Figure 3 further reports the distribution of selected rules during the adaptive

Algorithm 1: Adaptive mangling rules attack.

Data: dictonary D, rules-set R, budget β, neural net π_R

```
forall w ∈ D do
    R_w^β = {r | π_R(w)_r > (1 − β)};
    forall r ∈ R_w^β do
        g = r(w);
        issue g;
```



(a) *MyHeritage* on *animoto* (b) *animoto* on *MyHeritage* (c) *animoto* on *RockYou* (d) *RockYou* on *animoto*

Figure 2: Comparison between adaptive and classic mangling rules on four combination password leaks (dictionary/attacked-set) using the rules-set *PasswordPro*. $\beta=0.5$ is used for the adaptive case.

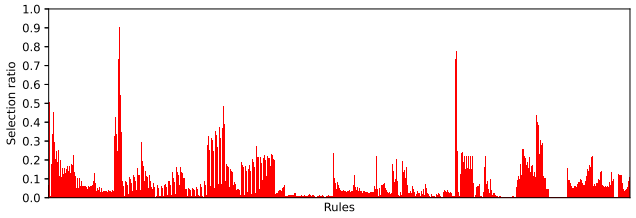


Figure 3: Selection frequencies of adaptive mangling rules for the 3120 rules of *PasswordPro*.

attack of Figure 2a. It emphasizes how mangling rules heterogeneously interact with the underlying dictionary. Although very few rules interact well with all the words (e.g., selection frequency is $> 70\%$), most of the mangling rules participate only in rare events.

Further empirical validation for the adaptive mangling rules will be given later in Section 5.

The Attack Budget Unlike standard dictionary attacks, whose effectiveness solely depends on the initial configuration, adaptive mangling rules can be controlled by an additional scalar parameter that we refer to as the **attack budget** β. This parameter defines the threshold of compatibility that a rule must exceed to be included in the rules-set R_w^β for a word w. Indirectly, this value determines the average size of compatible rules-sets, and consequently, the total number of guesses performed during the attack. More precisely, low values of β force compatible rule-sets to include only rules with high-compatibility. Those will produce only a limited number of guesses, inducing very precise attacks at the cost of missing possible hits (i.e., high precision, low recall). Higher values of β translate in a more permissive selection, where also rules with low-compatibility are included in the compatible set. Those will increase the number of produced guesses, inducing more exhaustive, yet more imprecise, attacks (i.e., higher recall, lower precision). When β reaches 1, the adaptive mangling rules attack becomes a standard mangling rules attack, since all the rules are unconditionally included in the compatible rules-set. The effect of the budget parameter is better captured by the examples reported in Figure 4. Here, the performance of multiple values of β is visualized and compared with the total hits and guesses performed by a standard mangling rules attack.

The budget parameter β can be used to model different types of adversaries. For instance, rational attackers [5] change their configuration in consideration of the practical cost of performing the attack. This parameter permit to easily describe those attackers and evaluate password security accordingly. For instance, using a low budget (e.g., $\beta=0.4$), we can model a greedy attacker who selects an attack configuration that maximizes guessing precision at the expense of the number of compromised accounts (a rational behavior in case of an expensive hash function).

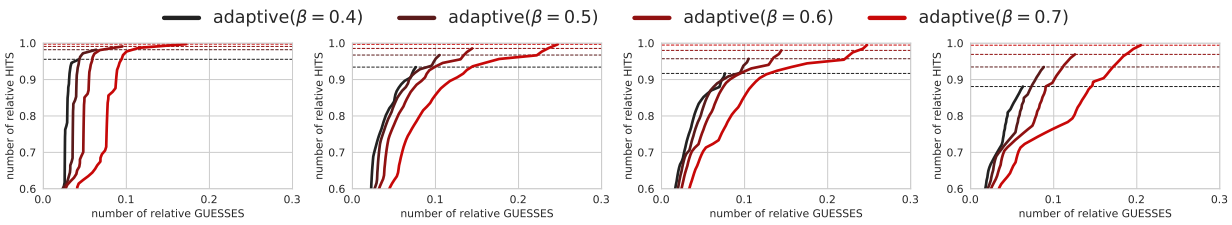
Seeking a more pragmatic interpretation, the budget parameter is implicitly equivalent to *early-stopping*[4] (i.e., Eq. 1), where single guesses are sorted in optimal order i.e., guesses are exhaustively generated before the attack, and indirectly sorted by decreasing probability/compatibility.

The optimal value of β depends on the rules-set. In our tests, we found these optimal values to be 0.6, 0.8 and 0.8 for *PassowordPro*, *generated* and *generated2*, respectively. Hereafter, we use these setups, unless otherwise specified.

Computational cost One of the core advantages of dictionary attacks over more sophisticated approaches [28, 31, 44] is their speed. For mangling rules attacks, generating guesses has almost a negligible impact. Despite being consistently more complex in their mechanisms, adaptive mangling rules do not tend to change this feature.

In Algorithm 1, the only additional operation over the standard mangling rules attack is the selection of compatible rules for each dictionary-word via the trained neural net. As discussed in Section 3.2.1, this operation requires just a single network inference to be computed; that is, with a single inference, we obtain a compatibility score for each element in $\{w\} \times R$. Furthermore, inference for multiple consecutive

[4] The attack stops before the guesses are terminated.



(a) *MyHeritage* on *animoto* (b) *animoto* on *MyHeritage* (c) *animoto* on *RockYou* (d) *RockYou* on *animoto*

Figure 4: Effect of the parameter β on the guessing performance for four different combinations of password sets and *PasswordPro* rules. Plots are normalized according to the results of the standard mangling rules attack (i.e., $\beta = 1$). For instance, $(x{=}0.1,\ y{=}0.95)$ means that we guessed 95% of the password guessed with the standard mangling rules attack by performing 10% of the guesses required from the latter.

words can be trivially batched and computed in parallel, further reducing the computation's impact.

Table 4 reports the number of compatibility values that different neural networks can compute per second. In the table, we used our largest networks without any form of optimization. Nevertheless, the overhead over the plain mangling rules attack is minimal (see Appendix D). Additionally, similar to standard dictionary attacks, adaptive mangling rules attacks are inherently parallel and, therefore, distributed and scalable.

4 Dynamic Dictionary attacks

This section introduces the second and last component of our password model—a dynamic mechanism that systematically adapts the guessing configuration to the unknown attacked-set. In Section 4.1, we introduce the **Dynamic Dictionary Augmentation** technique. Next, in Section 4.2, we introduce the concept of a **Dynamic Budgets**.

Motivation: As widely documented [6, 10, 27, 32], password composition habits slightly change from sub-population to sub-population. Although passwords tend to follow the same general distribution, credentials created under different environments exhibit unique biases. Users within the same group usually choose passwords related to each other, influenced mostly by environmental factors or the underlying applicative layer. Major factors, for example, are users' mother tongue [10], community interests [46] and, imposed password composition policies [23]. These have a significant impact on defining the final password distribution, and, consequently, the *guessability* of the passwords [21]. The same factors that shape a password distribution are generally available to the attackers who can collect and use them to drastically improve the configuration of their guessing attacks. Unfortunately, current automatic guessing techniques fail to describe this natural adversarial behavior [21, 26, 27, 41, 45]. Those methods are based on static configurations that apply the same guessing strategy to each attacked-set of passwords, mostly ignoring trivial information that can be either *a priori* collected or distilled from the running attack. In this section, we discuss suitable modifications of the mangling-rules framework to describe a more realistic guessing strategy. In particular, avoiding the necessity of any prior knowledge over the attacked-set, we rely on the concept of **dynamic attack** [32]. Here, a dynamic attacker is an adversary who changes his guessing strategy according to the attack's success rate. Successful guesses are used to select future attempts with the goal of exploiting the non-i.i.d. of passwords originated from the same environment. In other words, dynamic password guessing attacks automatically collect information on the target password distribution and use it to forge unique guessing configurations for the same set during the attack. Similarly, this general guessing approach can be easily linked to the optimal guessing strategy harnessed from human experts in [41], where mangling rules were manually created at execution time based on the initially guessed passwords.

Table 4: Number of compatible scores computed per second (c/s) for different networks. Values computed on a single NVIDIA V100 GPU.

generated2 *(large)*	**generated** *(large)*	**PasswordPro** *(large)*
130.550.403 c/s	89.049.382 c/s	31.836.734 c/s

4.1 Dynamic Dictionary Augmentation

In [32], dynamic adaptation of the guessing strategy is obtained from password latent space manipulations of deep generative models. A similar effect is reproduced within our mangling rules approach by relying on a consistently simpler, yet effective, solution based on *hits-recycling*. That is, every time we guess a new password by applying a mangling rule over a dictionary word, we insert the guessed password in the dictionary at runtime. In practice, **we dynamically augment the dictionary during the attack using the guessed pass-**

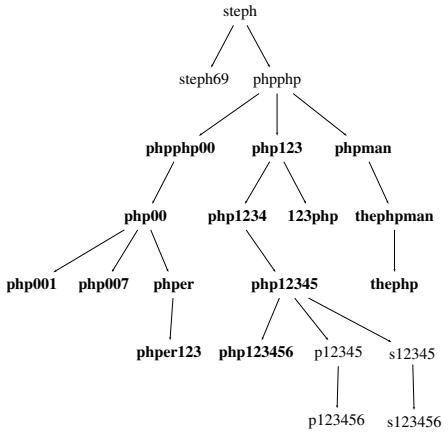


Figure 5: Example of small *hits-tree* induced by the dynamic attack performed on the *phpBB* leak. In the tree, every vertex is a guessed password; an edge between two nodes indicates that the child password has been guessed by applying a mangling rule to the parent password.

words.[5] In the process, every new hit is directly reconsidered and syntactically extended through mangling rules. This recursive method brings about massive chains/trees of hits that can extend for thousands of levels.[6] Figure 5 depicts an extremely small subtree ("*hits-tree*") obtained by attacking the password leak *phpBB*. The tree starts when the word "*steph*" is mangled, incidentally producing the word "*phpphp*". Since the latter lies in a dense zone of the attacked set (i.e., it is a common users' practice to insert the name of the website or related strings in their password), it induces multiple hits and causes the attack to focus in that specific zone of the key-space. The focus of the attack grows exponentially hit after hit and automatically stops only when no more passwords are matched. Eventually, this process makes it possible to guess passwords that would be missed with the static approach. For instance, in Figure 5, all the nodes in bold are passwords matched by the *dynamic* attack but missed by the *static* one (i.e., standard dictionary attack) under the same configuration.

Figure 6 compares the guessing performance of the dynamic attack against the static version on a few examples for the *PasswordPro* rules-set. The plots show that the dynamic augmentation of the dictionary has a very heterogeneous effect on the guessing attacks. In the case of Figure 6a, the dynamic attack produces a substantial increment in the number of guesses as well as in the number of hits i.e., from $\sim 15\%$ to $\sim 80\%$ recovered passwords. Arguably, such a gap is due to the minimal size of the original dictionary *phpBB*. In the attack of Figure 6b, instead, a similar improvement is achieved by requiring only a small number of guesses. On the other hand, in the attack depicted in Figure 6c, the dynamic augmentation has a limited effect on the final hits number. However, it increases the attack precision in the initial phase. Conversely, attacks in Figures 6d and 6e show a decreased precision in the initial phase of the attack, but that is compensated later by the dynamic approach.

Another interesting property of the dynamic augmentation is that it makes the guessing attack consistently less sensitive to the choice of the input dictionary. Indeed, in contrast with the static approach, different choices of the initial dictionary tend to produce very homogeneous results in the dynamic approach. This behavior is captured in Figure 7, where results, obtained by varying three input dictionaries, are compared between static and dynamic attack. The standard attacks (Figure 7a) result in very different outcomes; for instance, using *phpBB* we match 15% of the attacked-set, whereas we match more than 80% with *MyHeritage*. These differences in performance are leveled out by the dynamic augmentation of the dictionary (Figure 7b); all the dynamic attacks recover $\sim 80\%$ of the attacked-set. Intuitively, dynamic augmentation remedies deficiencies in the initial configuration of the dictionary, promoting its completeness. These claims will find further support in Section 5.

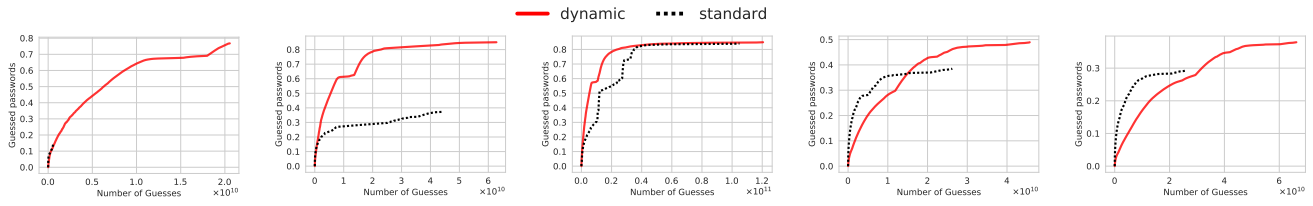
4.2 Dynamic budgets

Adaptive mangling rules (Section 3.3) demonstrated that it is possible to consistently improve the precision of the guessing attack by promoting compatibility among rules-set and dictionary (i.e., simulating high-quality configurations at runtime). This approach assumes that the compatibility function modeled before the attack is sufficiently general to simulate good configurations for each possible attacked-set. However, as motivated in the introduction of Section 4, every attacked set of passwords present peculiar biases and, therefore, different compatibility relations among rules and dictionary-words.
To reduce the effect of this dependence, we introduce an additional dynamic approach supporting the adaptive mangling rules framework. Rather than modifying the neural network at runtime (which is neither a practical nor a reliable solution), we alter the selection process of compatible rules by acting on the budget parameter β.

Algorithm 2 details our solution. Here, rather than having a global parameter β for all the rules of the rules-set R, we have a budget vector B that assigns a dedicated budget value to each rule in R (i.e., $B \in (0,1]^{|R|}$). Initially, all the budget values in B are initialized to the same value β (i.e., $\forall_{r \in R}\, B_r = \beta$) given as an input parameter. During the attack, the elements of B are individually increased and decreased to better describe the attacked set of passwords. Within this context, increasing the budget B_r of a rule r means reducing the compatibility threshold needed to include r in the compatible rules-set of a dictionary-word w, and, consequently, making r more popular

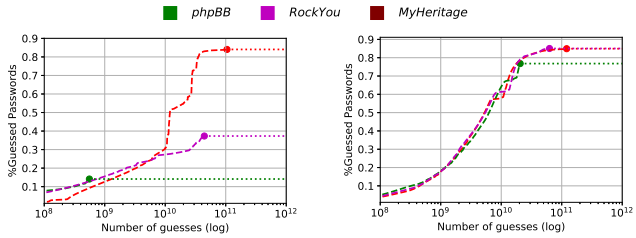
[5] Although we have not found any direct reference to the *hits-recycling* technique in the literature, it is likely well known and routinely deployed by professionals.

[6] I.e., a forest, where the root of each tree is a word from the original dictionary.



(a) *phpBB* on *animoto* (b) *RockYou* on *animoto* (c) *MyHeritage* on *animoto* (d) *animoto* on *RockYou* (e) *animoto* on *MyHeritage*

Figure 6: Performance comparison between dynamic and standard (static) attack for five different setups of dictionary/attacked-set. The rules set *PasswordPro* in non-adaptive mode is used in all the reported attacks. The 5 setups have been handpicked to fully represent the possible effects of the dynamic dictionary augmentation.



(a) standard attack (b) dynamic attack

Figure 7: Guessing attacks performed on the *animoto* leak using three different dictionaries. The panel on the left reports the guessing curves for the static setup. The panel on the right reports those for the dynamic setup. The x-axis is logarithmic.

during the attack. On the other hand, by decreasing B_r, we reduce the chances of selection for r; r is selected only in case of high-compatibility words.

In the algorithm, we increase the budget B_r when the rule r produces a hit. The added increment is a small value Δ that scales inversely with the number of guesses produced. At the end of the internal loop, the vector B is then normalized; i.e., we scale the values in B so that $\sum_r^R B_r = \sum_i^{|R|} \beta$. Normalizing B has two aims. **(1)** It reduces the budgets for non-hitting rules (the mass we add to the budget of rule r is subtracted from all other budgets). **(2)** It maintains the total budget of the attack (i.e., $\sum_i^{|R|} \beta$) unchanged so that dynamic and static budget leads to almost the same number of guesses during the attack for a given β. Furthermore, we impose a maximum and a minimum bound on the increments or decrements of B. This is to prevent values of zero (rule always excluded) or equal/higher than one (rule always included).

Algorithm 2: Adaptive rules with Dynamic budget

Data: dictonary D, rules-set R, attacked-set X, budget β

forall $w \in D$ **do**
  $R_w^\beta = \{r | \pi_R(w)_r > (1 - B_i)\}$;
  forall $r \in R_w^\beta$ **do**
    $g = r(w)$;
    if $g \in X$ **then**
      $X = X - \{g\}$;
      $B_r = B_r + \Delta$;
    $B = B \cdot \frac{\sum^{|B|} \beta}{\sum^{|B|} B}$;

As for the dynamic dictionary augmentation, the dynamic budget has always a positive, but, heterogeneous, effect on the guessing performance. Mostly, the number of hits increases or remains unaffected. Among the proposed techniques, this is the one with the mildest effect. Yet, this will be particularly useful when combined with dynamic dictionary augmentation in the next section. Appendix C better explicates the improvement induced from the dynamic budgets.

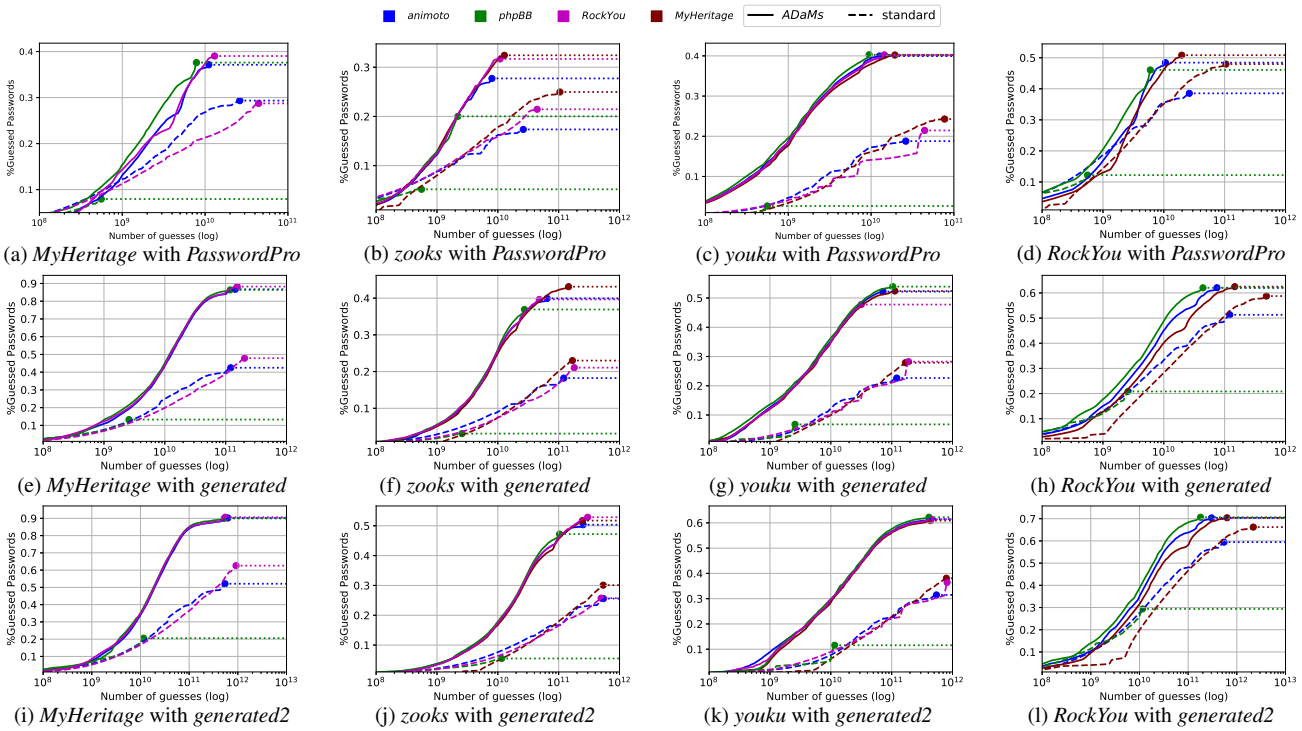
5 Adaptive, Dynamic Mangling rules: *AdaMs*

The results of the previous section confirm the effectiveness of the dynamic guessing mechanisms. We increased the number of hits compared to classic dictionary attacks by using the produced guesses to improve the attack on the fly. However, in the process, we also increased the number of guesses, possibly in a way that is hard to control and gauge. Moreover, by changing the dictionary at runtime, we disrupt any form of optimization of the initial configuration, such as any *a priori* ordering in the wordlist [26] and any joint optimization with the rules-set[7]. Unavoidably, this leads to sub-optimal attacks that may overestimate passwords strength.

To mitigate this phenomenon, we combine the dynamic augmentation technique with the complementary Adaptive Mangling Rules framework. The latter seeks an optimal configuration at runtime on the dynamic dictionary, promoting compatibility with the rules-set and limiting the impact of imperfect dictionary-words even if these are unknown before the attack. This process is further supported by the dynamic budgets that address the possible covariate-shift [40] of the compatibility function induced by the augmented dictionary.

Hereafter, we refer to this final guessing strategy as *AdaMs* (**A**daptive, **Dyn**amic **M**angling rule**s**). Details on the

[7] I.e., new words may not interact well with the mangling rules in use.



(a) *MyHeritage* with *PasswordPro* (b) *zooks* with *PasswordPro* (c) *youku* with *PasswordPro* (d) *RockYou* with *PasswordPro*

(e) *MyHeritage* with *generated* (f) *zooks* with *generated* (g) *youku* with *generated* (h) *RockYou* with *generated*

(i) *MyHeritage* with *generated2* (j) *zooks* with *generated2* (k) *youku* with *generated2* (l) *RockYou* with *generated2*

Figure 8: Each plot reports the number of guesses (in log scale) and the percentage of matched passwords for different rule-sets and dictionaries against several attacked-sets. Each row reports a rule-set, whereas each column identifies an attacked-set. We use four dictionaries, each identified by a colored line. Continuous lines show *AdaMs* attacks whereas dashed lines refer to standard mangling rules attacks.

implementation of *AdaMs* are given in Appendix E, whereas we benchmark it in Appendix D.

5.1 Evaluation

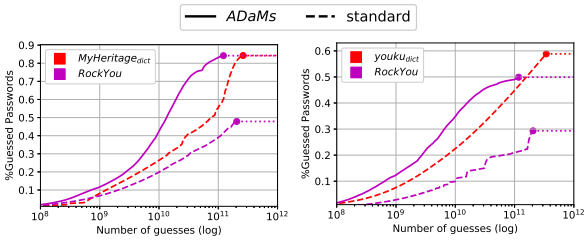
Figure 8 reports an extensive comparison of *AdaMs* against standard mangling-rules attacks. In the figure, we test all pairs of dictionary/rule-set obtained from the combination of the dictionaries: *MyHeritage*, *RockYou*, *animoto*, *phpBB* and the rules-sets: *PasswordPro*, *generated* and *generated2* on four attacked-sets. Hereafter, we switch to a logarithm scale given the heterogeneity of the number of guesses produced by the various configurations.

For the reasons given in the previous sections, *AdaMs* outperforms standard mangling rules within the same configurations, while requiring fewer guesses on average. More interestingly, *AdaMs* attacks generally exceed the hits count of all the standard attacks regardless of the selected dictionary. In particular, this is always true for the *generated* and *generated2* rules-sets.

Conversely, in cases where the dynamic dictionary augmentation offers only a small gain in the number of hits (e.g., attacking *RockYou*), *AdaMs* equalizes the performance of various dictionaries, typically, towards the best configuration for the standard attack. In Figures 8d and 8h, all the configurations of *AdaMs* reach a number of hits comparable to the best configuration for the standard attack, i.e., using *MyHeritage*, while requiring up to an order of magnitude fewer guesses (e.g., Figure 8d), further confirming that the best standard attack is far from being optimal. In the reported experiments, the only outlier is *phpBB* when used against *zooks* in Figure 8b. Here, *AdaMs* did not reach/exceed all the standard attacks in the number of hits despite consistently redressing the initial configuration. However, this discrepancy is canceled out when more mangling rules are considered such as in Figure 8f.

Eventually, the *AdaMs* attack makes the initial selection of the dictionary systematically less influential. For instance, in our experiments, a set such as *phpBB* reaches the same performance of wordlists that are two orders of magnitude larger (e.g., *RockYou*). The crucial factor remains the rules-set's **cardinality** that ultimately determines the magnitude of the attack, even though it does not appreciably affect the guessing performance.

The effectiveness of *AdaMs* is better captured by the results reported in Figure 9. Here, we create a synthetic optimal dictionary for an attacked-set and evaluate the capability of *AdaMs* to converge to the performance of such an optimal configuration. To this end, given a password leak X, we ran-



(a) $MyHeritage_{\text{target}}$ (b) $youku_{\text{target}}$

Figure 9: Comparison of *AdaMs* against optimal dictionary for two sets of passwords.

domly divide it in two disjointed sets of equal size, say X_{dict} and X_{target}. Then, we attack X_{target} by using both X_{dict} (i.e., optimal dictionary) and an external dictionary (i.e., sub-optimal dictionary). Arguably, X_{dict} is the *a priori* optimal dictionary to attack X_{target} since X_{dict} and X_{target} are samples of the very same distribution.

We report the results for two sets: *MyHeritage* and *youku*. The attacks are carried out by using the rules-set *generated* and *RockYou* as the external dictionary. In the case of *MyHeritage*, the *AdaMs* attack is more precise than the optimal dictionary and produces a comparable number of hits. Similarly, in the case of *youku*, the *AdaMs* attack guesses faster than the optimal dictionary within the first 10^{11} guesses. However, in this case, it does not reach an equivalent number of guessed passwords. We can attribute this to the high discrepancy between the initial dictionary *RockYou* and the attacked-set *youku* that cannot be bridged without prior knowledge.[8] Nevertheless, the dictionary augmentation technique can induce a dictionary that has a comparable utility to one of the best optimal *a priori* setup, while requiring no information on the attacked-set. In the process, the adaptive framework consistently accounts for the noise introduced by the augmentation, allowing *AdaMs* to be even more precise than the optimal dictionary for most of the attack (i.e., within the first 10^{11} guesses).

Further comparison with other password models can be found in Appendix A.

6 Takeaways and New Directions

The *AdaMs* attack autonomously pushes the attack strategy towards the optimal one, producing password strength estimates that better model actual adversarial capabilities. As shown in Figure 8, the approach also makes the guessing attack more resilient to deficiencies in the initial configuration, reducing the bias induced by misconfiguration. In this direction, the *AdaMs* attack further proves the intrinsic unsuitability of arbitrarily chosen configurations and the overestimation of password security that those can induce.

Compared with other systems [28, 32], our framework provides researchers and security practitioners with a markedly more efficient and flexible solution. We make our code and trained models publicly available[9] in the hope **our system will help improve the soundness of password strength estimation techniques**.

Finally, our techniques pave the way for new valuable directions in the study of password security: **(1)** our dynamic attack offers a framework capable of explaining causality relations among guessed passwords in a dynamic context; the *hits-tree* produce from our technique could provide insights on how to proactively reduce the threat of dynamic attackers. **(2)** Mangling rules are not necessarily *effective* or *ineffective* as assumed in current automatic configuration techniques [4, 26]. They have a conditional nature that must be accounted for to seek optimal configurations. Adaptive mangling rules have proven to be superior and more effective. Still, it would be interesting to devise new techniques to automatically formulate mangling rules rather than select and compose existing ones.

Acknowledgements

We wish to thank Blase Ur (our *shepherd*) and the other anonymous reviewers for the valuable feedback which helped to improve the paper.

References

[1] "Over 1.4 Billion Clear Text Login Credentials Found in Single Database on Dark Web". https://infowatch.com/analytics/leaks_monitoring/97798.

[2] "techcrunch.com: Animoto hack exposes personal information, location data". https://tinyurl.com/ybhc9uaz.

[3] "arstechnica.com: Anatomy of a hack: How crackers ransack passwords like "qeadzcwrsfxv1331". https://tinyurl.com/y9jw4va6.

[4] "Automatic mangling rules generation". http://passwords12.at.ifi.uio.no/Simon_Marechal-manglingrules_Passwords12.pdf.

[5] J. Blocki, B. Harsha, and S. Zhou. On the economics of offline password cracking. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 853–871, 2018.

[6] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012*

[8] The leak *youku* is mostly composed of Chinese passwords that are underrepresented in *RockYou*.

[9] https://github.com/TheAdamProject/adams

IEEE Symposium on Security and Privacy, pages 538–552, 2012.

[7] S. Boztas. Entropies, guessing and cryptography, 1999.

[8] "Crack me if you can contest". https://contest.korelogic.com.

[9] X. de Carné de Carnavalet and M. Mannan. From very weak to very strong: Analyzing password-strength meters. In *NDSS*, 01 2014.

[10] M. Dell' Amico, P. Michiardi, and Y. Roudier. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[11] M. Dell'Amico and M. Filippone. Monte carlo strength evaluation: Fast and reliable password checking. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 158–169, New York, NY, USA, 2015. Association for Computing Machinery.

[12] S. Fahl, M. Harbach, Y. Acar, and M. Smith. On the ecological validity of a password study. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, New York, NY, USA, 2013. Association for Computing Machinery.

[13] "neural_network_cracking: Github repository". https://github.com/cupslab/neural_network_cracking.

[14] A. Forget, S. Chiasson, P. C. van Oorschot, and R. Biddle. Improving text passwords through persuasion. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, SOUPS '08, page 1–12, New York, NY, USA, 2008. Association for Computing Machinery.

[15] "hashcat - advanced password recovery". https://hashcat.net/hashcat.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[17] M. Jakobsson and M. Dhiman. The benefits of understanding passwords. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security*, HotSec'12, page 10, USA, 2012. USENIX Association.

[18] "John the Ripper password cracker". https://www.openwall.com/john.

[19] "John's Markov generator". https://openwall.info/wiki/john/markov.

[20] B. Kaliski. "Pkcs# 5: Password-based cryptography specification version 2.0". https://tools.ietf.org/html/rfc2898, 2000.

[21] 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 *2012 IEEE Symposium on Security and Privacy*, pages 523–537, 2012.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[23] 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 *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, New York, NY, USA, 2011. ACM.

[24] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[25] "LinkedIn Hack Wikipedia". https://en.wikipedia.org/wiki/2012_LinkedIn_hack.

[26] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur. Reasoning Analytically About Password-Cracking Software. In *IEEE Symposium on Security and Privacy*, SP '19, pages 1272–1289, San Francisco, California, USA, May 2019. IEEE.

[27] M. L. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, page 173–186, New York, NY, USA, 2013. Association for Computing Machinery.

[28] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 175–191, Austin, TX, Aug 2016. USENIX Association.

[29] R. Morris and K. Thompson. Password security: A case history. *Commun. ACM*, 22(11):594–597, November 1979.

[30] "The Verge: MyHeritage breach leaks millions of account details". https://tinyurl.com/y7w6wsrf.

[31] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[32] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti. Improving Password Guessing via Representation Learning. In *IEEE Symposium on Security and Privacy (SP)*, 2021.

[33] "pcfg_cracker: PCFG Github repository". https://github.com/lakiw/pcfg_cracker.

[34] C. Percival. Stronger key derivation via sequential memory-hard functions, 01 2009.

[35] N. Provos and D. Mazières. A future-adaptive password scheme. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '99, page 32, USA, 1999. USENIX Association.

[36] "computerworld.com: RockYou hack exposes names, passwords of 30M accounts'. https://tinyurl.com/yyn87l48.

[37] "I have the HashCat so I make the rules". https://hashcat.net/events/p14-vegas/I%20have%20the%20%23cat%20i%20make%20the%20rules_YC.pdf.

[38] S. Schechter, C. Herley, and M. Mitzenmacher. 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'10, page 1–8, USA, 2010. USENIX Association.

[39] D. Seeley. Password cracking: A game of wits. *Commun. ACM*, 32(6):700–703, June 1989.

[40] M. Sugiyama, M. Krauledat, and K. R. Müller. Covariate shift adaptation by importance weighted cross validation. *J. Mach. Learn. Res.*, 8:985–1005, December 2007.

[41] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay. Measuring real-world accuracies and biases in modeling password guessability. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 463–481, Washington, D.C., Aug 2015. USENIX Association.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[43] R. Veras, C. Collins, and J. Thorpe. On the semantic patterns of passwords and their security impact. ndss, 2014.

[44] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405, 2009.

[45] D. L. Wheeler. zxcvbn: Low-budget password strength estimation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, Austin, TX, Aug 2016. USENIX Association.

[46] R. V. Yampolskiy. Analyzing user password selection behavior for reduction of password space. In *Proceedings 40th Annual 2006 International Carnahan Conference on Security Technology*, pages 109–115, 2006.

[47] C. Yeh, W. Wu, W. Ko, and Y. Wang. Learning deep latent spaces for multi-label classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2838–2844. AAAI Press, 2017.

[48] "HackRead: Chinese Video Service Giant Youku Hacked; 100M Accounts Sold on Dark Web". https://tinyurl.com/yb78uxnh.

[49] "The Economic Times: Zomato hacked: Security breach results in 17 million user data stolen". https://tinyurl.com/y8xec7sr.

[50] "arstechnica.com: Dating site Zoosk resets some user accounts following password dump". https://tinyurl.com/y3r2xob5.

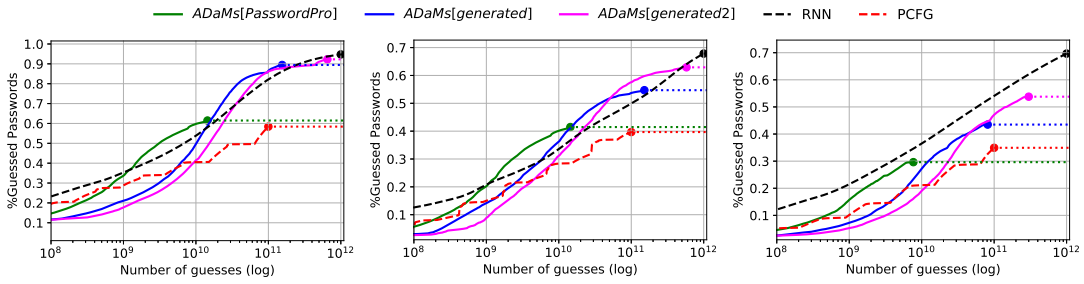
Appendices

A Comparison with other password models

Next, we compare *AdaMs* with other password models.

Figure A.1 reports a direct comparison against the RNN-based approach of Melicher et al. [28] and PCFG [44]. The RNN-based password model is the state-of-the-art for password strength estimation, although its computational cost in generating guesses makes it impractical for real password guessing. We train the model using *RockYou* and simulate password guessing attacks using [11]. In the process, we use default parameters of the available software [13] and consider passwords with guess-number within 10^{12}.

PCFG is the academic approach that better mirrors the guessing generation process of dictionary attacks. We train the

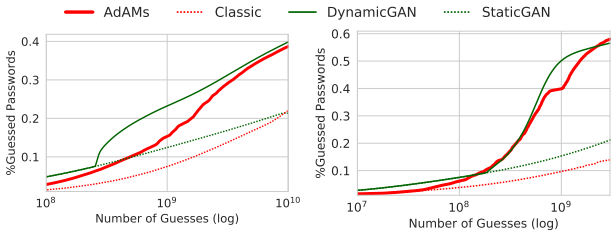


(a) *MyHeritage* (b) *youku* (c) *zooks*

Figure A.1: Comparison of the *AdaMs* attacks against the RNN-based approach of Melicher et al. [28] and PCFG [44] for three password leaks.

PCFG-based model on *RockYou* using the default setting [33]. In this case, we limit to the first 10^{11}.

We compare the models on three leaks: *MyHeritage*, *youku* and *zooks*. For the *AdaMs* attacks, we use *RockYou* as a dictionary, whereas we report results for three rules-sets. Surprisingly, the *AdaMs*reach performance very close to the one obtained from the RNN-based model. It even outperforms the parametric attack in two of the three attack-sets. Similarly, *AdaMs* tend to perform better than PCFG in the three cases, especially after the initial guesses. Furthermore,



(a) *youku* (b) *zomato*

Figure A.2: Performance comparison between *AdaMs* and the dynamic attack [32]. Classic mangling rules attacks and StaticGAN [32] are reported as baseline.

Figure A.2 compares *AdaMs* against the original GAN-based dynamic attack [32]. We base the comparison on the same leaks used in [32]; namely, the *youku* and *zomato* leak (details given in Table 2). The GAN-based model is trained on the *RockYou* leak and the attack is performed with the same hyperparameters used in [32]: $\sigma = 0.35$ and *hot-start* $\alpha = 10\%$. Despite our simpler approach, the *AdaMs* attack performs very similarly to the GAN-based attack, besides being significantly faster in generating guesses (see Table D.1).

B Details on the deep learning framework

This Appendix details the architecture used to implement the neural approximations of the compatibility functions presented in Section 3.2.1. It can be defined using five parameters, namely:

Algorithm 3: Residual Block: `residualBlock(·)`:

Data: input tensor: x_{in}
1 $x = \texttt{batchNormalization}(x_{in})$;
2 $x = \texttt{ReLU}(x)$;
3 $x = \texttt{1D-Convolution}(x, f, k)$;
4 $x = \texttt{batchNormalization}(x)$;
5 $x = \texttt{ReLU}(x)$;
6 $x = \texttt{1D-Convolution}(x, f, k)$;
7 **return** $x_{in} + 0.3 \cdot x$

Algorithm 4: Architecture:

Data: input tensor: x_{in}, rules-set R
1 $x = \texttt{charactersEmbedding}(x_{in}, 128)$;
2 $x = \texttt{1D-Convolution}(x, f, k)$;
3 **for** 0 ***to*** d **do**
4 $\quad x = \texttt{residualBlock(x)}$
5 $bneck = \lceil \frac{f}{b} \rceil$;
6 $x = \texttt{1D-Convolution}(x, bneck, k)$;
7 $x = \texttt{flattern}(x)$;
8 $logits = \texttt{dense}(x, |R|)$;
9 **return** *logits*

- **Depth (d):** The number of residual blocks composing the network. Each residual block includes two 1D-convolutional layers, supported by normalization layers and activation i.e., Algorithm 3.
- **Number of filters (f):** The number of filters for each convolutional layer in the network.
- **Kernel size (k):** Size of the kernel used in every convolutional layer in the network.
- **Final Bottleneck (b):** Reduction of the number of filters before the final dense layer.

The final architecture is described in Algorithm 4. Our biggest models are realizations of the parameters: $d=15$, $f=512$, $k=5$. We use $b=2$ for *PasswordPro* and *generated*, $b=3$ for *generated2* instead.

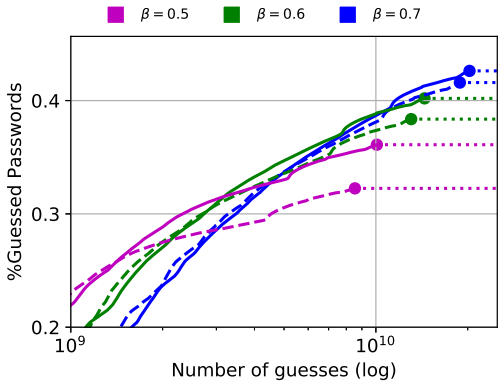


Figure C.1: Effectiveness of the dynamic-budget within *AdaMs* for different value of β. Continuous lines present *AdaMs*, whereas dashed lines are *AdaMs* ablated of the dynamic-budget

Table D.1: Number of guesses per second compute single core/GPU on a NVIDIA DGX-2 machine.

AdaMs **generated2**	*AdaMs* **generated**	*AdaMs* **PasswordPro**	**Hashcat** CPU legacy	**GAN [32]** Dynamic Attack
726182 g/s	709439 g/s	644444 g/s	928647 g/s	34189 g/s

C Impact of the Dynamic budget

We briefly illustrate the impact of the dynamic budget (i.e., Section 4.2) on the performance of *AdaMs*. As previously discussed, the dynamic budget has always a positive or neutral effect. Figure C.1 reports an example for the attacked-set *youku*. In the figure, continuous lines refer to the complete *AdaMs* attack, whereas dashed lines report the results for *AdaMs* without dynamic budget for the same configuration. We report the results for three values of β.
As shown in the example, the dynamic budget is particularly effective when low β is used. In these cases, the dynamic logic helps better organize the small total budget of the attack, resulting in better global performance. The gain decreases when bigger budgets are adopted.

D Benchmarks

In this Appendix, we analyze the computational cost of generating guesses with *AdaMs*. Primarily, we test the overhead with respect to standard mangling rules (i.e., *Hashcat CPU legacy*).

For the comparison, we produce 10^9 strings and compute the number of guesses generated per second (i.e., g/s). **In the process, we include the time of checking for the guesses in the set of the attacked passwords** (the same methodology is used for each tool and may not be computationally optimal). Note that we do not perform any hash function computation in the process. We repeat the test 5 times using *RockYou* as dictionary and *animoto* as attacked-set, whereas we repeat for the rules-sets: *PasswordPro*, *generated* and *generated2*. Table D.1 averages the time for each tool. The result for the standard mangling rules is reported as average over the three rules-sets. Additionally, we report the timings for the GAN-based, dynamic attack described in [32].

On average, *AdaMs* are just 25% slower than standard mangling rules. Considering that the Adaptive mangling rules can reduce the number of guesses up to an order of magnitude, this overhead becomes negligible in practice. Moreover, this discrepancy easily fades out when slow hash functions, such as [20, 34, 35], are considered.

E Implementation of *AdaMs*

We rely on the *CPU legacy* version of *Hashcat*[10] to implement *AdaMs* attacks. Our prototype uses the CPU version as it is easier to modify its workflow, although the *Hashcat* GPU engine can trivially support our approach.[11]
In the code, we modify the main loop of *Hashcat*, where it scans over dictionary words and then iterates on all rules. We read a batch of words from the dictionary, we give them as input to the neural network, and then, for each word w in the batch, we apply only the rules whose values of α_R are greater than $(1-\beta)$. We check all these guesses and, those who match are added on top of the remaining words in the dictionary, i.e., they will be part of the next batch of words. The same batching approach is used for the dynamic budget. Here, budget increments and normalization per rule are performed conjointly after every batch to further reduce computational overhead. In the implementation, we use batch-size equals to 4096 dictionary words.

[10] https://github.com/hashcat/hashcat-legacy
[11] The GPU engine is also more suited as it would naturally support the computation of the neural network on GPU, removing the CPU/GPU communication overhead