The Impact of Exposed Passwords on Honeyword Efficacy

Zonghao Huang
Duke University

Lujo Bauer
Carnegie Mellon University

Michael K. Reiter
Duke University

Abstract

Honeywords are decoy passwords that can be added to a credential database; if a login attempt uses a honeyword, this indicates that the site's credential database has been leaked. In this paper we explore the basic requirements for honeywords to be effective, in a threat model where the attacker knows passwords for the same users at other sites. First, we show that for user-chosen (vs. algorithmically generated, i.e., by a password manager) passwords, existing honeyword-generation algorithms do not simultaneously achieve false-positive and false-negative rates near their ideals of ≈ 0 and $\approx \frac{1}{1+n}$, respectively, in this threat model, where n is the number of honeywords per account. Second, we show that for users leveraging algorithmically generated passwords, state-of-the-art methods for honeyword generation will produce honeywords that are not sufficiently deceptive, yielding many false negatives. Instead, we find that only a honeyword-generation algorithm that uses the *same* password generator as the user can provide deceptive honeywords in this case. However, when the defender's ability to infer the generator from the (one) account password is less accurate than the attacker's ability to infer the generator from potentially many, this deception can again wane. Taken together, our results provide a cautionary note for the state of honeyword research and pose new challenges to the field.

1 Introduction

Credential database breaches have long been a widespread security problem and are only becoming more so. The 2022 Verizon Data Breach Investigations Report places credentials as one of the two most often breached types of confidential data, since they are so useful for attackers to masquerade as legitimate users on the system [43, p. 18]. Credential database breaches are the largest source of compromised passwords used in credential stuffing campaigns [38]. In turn, credential stuffing campaigns are the cause of the vast majority of account takeovers [35]. Unfortunately, there is usually a significant delay between the breach of a credential database and its discovery. Estimates of the average delay to detect a data breach range from six to eight months in a 2022 report [21, Fig. 45]. The resulting window of vulnerability gives attackers the opportunity to crack the passwords offline, and then sell them or leverage them directly [35, 38].

A strategy to accelerate the detection of credential database breaches, suggested by Juels and Rivest nearly a decade ago [22], is for a site to store decoy passwords, or *honeywords*, alongside real passwords in its credential database, so that if the attacker breaches the database, the correct passwords are hidden among the honeywords. The entry of a honeyword in a login attempt then alerts the site to its breach, since the legitimate user does not know the honeyword. In the time since this proposal, researchers have proposed various algorithms for generating honeywords (see Sec. 3) to meet two central criteria: (i) that it be difficult for an attacker who has breached a site's credential database to distinguish the legitimate password for an account from that account's honeywords, and (ii) that it be difficult for an attacker who has not breached a site's credential database to guess honeywords for an account, since such guesses will induce false breach alarms.

The tendency of users to reuse passwords across sites (e.g., [31, 54]) presents a challenge for honeywords, since an attacker can stuff an account's breached passwords at *other* sites where the same user has accounts, thereby discovering the legitimate password as the one that works at another site. As such, previous advances in honeyword system designs [52] provide a mechanism by which one site can monitor for the entry of another site's honeywords in local login attempts. Still, however, for an account for which the attacker can obtain the user's passwords on other sites (e.g., by breaching these other sites, or by phishing their passwords), the attacker will likely need not resort to credential stuffing to differentiate the legitimate password from its honeywords. While this might seem like an unnecessarily challenging threat model, it is unfortunately realistic: A July 2020 report found more than 15 billion credentials in circulation in cybercriminal marketplaces [11], or an average of more than two for every person on the planet.

In this paper we conduct the first critical analysis of honeyword-generation algorithms in this setting, i.e., wherein the attacker knows legitimate passwords at other sites for the users represented in a database it is targeting. There is reason to suspect that this threat model would pose significant challenges to honeyword efficacy for user-chosen (versus algorithmically generated) passwords. On the one hand, if the honeyword-generation algorithm used to populate the targeted database generates honeywords that are all dissimilar from the user-chosen password, then the known password(s) for the same user might enable the attacker to distinguish the user-chosen password from its honeywords with high probability. If so, the false-negative probability (the probability that the site fails to detect the breach) would be high. On the other hand, if the honeyword-generation algorithm generates some honeywords that are similar to the user-chosen password, then this might make it easier for an attacker who has *not* breached the database to guess and enter honeywords in login attempts, thereby inducing a false breach alarm (false positive).

Through a systematic analysis of current honeyword-generation algorithms, we quantify this tension and, by doing so, show that there appears to be no known algorithm providing a good tradeoff for accounts with user-chosen passwords. We additionally applied two password tweaking techniques from password guessing to improve honeyword generation. While these two algorithms relieve this tension by providing slightly lower false-negative probability, they still induce a high false-positive probability. Therefore, it remains far from clear that there is *any* honeyword-generation algorithm that ensures low false-negative probability and provides adequate resistance to false breach alarms (i.e., a false-positive rate near 0).

We then turn our attention to accounts with algorithmically generated passwords, as might be generated by a password manager. The critical finding that we uncover in this case is that honeyword-generation algorithms that do not take into account the method by which the legitimate password was generated will yield high false-negative probability. For example, if the user employs a password manager that generates passwords to fit a user-configured specification, and if the passwords exposed for that user permit the attacker to infer this specification, then the attacker can discard any honeywords not fitting that pattern. We will quantify the ability of the adversary to do so against existing honeyword-generation algorithms, most of which do not guarantee honeywords of the same pattern as the legitimate password. We then consider the possibility that the honeyword-generation algorithm itself leverages a password manager to generate honeywords whenever the user does. However, due to the numerous generator configurations that users might adopt, doing so is not foolproof. In particular, if the attacker knows potentially more passwords for the same user's accounts elsewhere, it can classify the user's typical configuration better than the defender can. This advantage thus implies an increase in false negatives, which we will demonstrate in certain cases.

To summarize, our contributions are as follows:

- We formalize the false-positive and false-negative rates of honeywords in a model in which the attacker possesses passwords for the same user at other sites (obtained by, e.g., breaching those sites or phishing the user).
- Using these definitions and empirical datasets of compromised passwords, we show that existing honeyword-generation algorithms (and two honeyword-generation methods adapted from password-guessing attacks) exhibit poor tradeoffs between false negatives and false positives in this threat model. All the analyzed methods have a false-negative rate much higher than random guessing (i.e., it is often easy for false-negative attackers to distinguish the account password from honeywords) or a false-positive rate much higher than zero (i.e., it is often easy for false-positive attackers to induce false breach alarms).
- We conduct the first study of using honeywords to protect algorithmically generated passwords. Though relevant only for sites that reversibly encrypt their password databases (since password hashing, which is best practice, should render algorithmically generated passwords irrecoverable to an attacker who breaches the database), our study provides interesting findings in this setting. Using passwords gathered from popular password managers, we show that introducing honeywords without attention to the account's password being algorithmically generated offers little protection for existing honeyword-generation algorithms. We further explore the use of automatic password generators to generate honeywords when the account password is identified as being algorithmically generated itself, but find that the myriad configurations of these generators can be a pitfall for honeyword generation.

Due to space constraints, some of our results are detailed only in the full paper [20]. Source code for conducting our experiments is available from `https://github.com/zonghaohuang007/honeywords-analysis`.

2 Related Work

2.1 Honeywords

Since honeywords were first proposed [22], there have been several research efforts on designing honeyword-generation techniques [1,6,13,15,48] or evaluating their security, mostly against attacks trying to access a breached site's accounts without alerting the site to its breach. In their original proposal, Juels and Rivest defined an abstract model of a honeyword system and proposed several legacy-UI methods including *chaffing-by-tweaking* and *chaffing-with-a-password-model (modeling syntax)*, and one modified-UI method. The modified-UI method requires the authentication system to guide the user in the selection of her account password and thus has inherent usability challenges, and so we do not consider it in this paper. The legacy-UI methods use random replacement of characters in the account password. We use one of them in this paper to represent this class of techniques, as discussed in Sec. 3. We also consider a method, called the "List" model in Sec. 3, that utilizes existing passwords as the honeywords for the site's accounts [46] (similar to Erguler [15]). A proposal by Dionysiou, et al. [13] leverages a machine learning model to search for similar passwords in the system and then generates n honeywords by tweaking the searched passwords randomly (e.g., by the chaffing-by-tweaking method), also described in Sec. 3. More recently, Yu and Martin [58] proposed to leverage the Generative Pre-trained Transformer 3 Model (GPT-3) [4] to generate honeywords. Their method includes two steps: first, a password-specific segmentation technique called PwdSegment [56] is used to extract chunks from the input password, and second, a prompt including the chunk information is provided as the input to GPT-3, which returns a list of passwords similar to the input password, used as honeywords. Wang and Reiter [53] proposed a honeyword-selection mechanism based on a Bernoulli selection that achieves tunable false positives.

Recent works have investigated the security of honeywords under *targeted-guessing attacks* where the attacker has personal identity information (PII) about the users. Wang, et al. [46] performed the first security analysis on honeywords under such attacks, but they focused only on the legacy-UI methods proposed by Juels and Rivest, empirically showing that these methods fail to achieve low false-negative rates. More recently, Wang, et al. [48] considered both PII and registration order (the time when the user accounts were created) as the auxiliary information available to the attacker. They proposed leveraging this auxiliary information in a password model like the List model, probabilistic context-free grammars (PCFG) [8], a Markov model [33], or a combination thereof, to generate honeywords. Their proposed methods achieved low false-negative rates under the threat model considered in their work [48]. However, our empirical results demonstrate that existing honeyword-generation techniques, including those considered by Wang, et al., have a high false-negative probability in our threat model. Setting a larger number of honeywords per account, as suggested by Wang, et al., generally lowers false-negative rates but increases false-positive rates. We are the first to systematically analyze the trade-off, showing that existing honeyword-generation methods suffer from high false-positive or false-negative rates under a threat model where the passwords of the same user from the other sites are exposed to attackers.

2.2 Password Guessing

A related topic to honeyword generation is password guessing, which is used to crack passwords [14, 55, 57] in an online or offline manner or used to evaluate their strength [10, 17, 23]. Since a honeyword is simply a decoy password, it is reasonable that honeyword research will benefit from the development of password guessing techniques. Weir, et al. [55] proposed the first method to utilize a probabilistic model to generate passwords. They designed the model using PCFGs trained on a training set of passwords and empirically demonstrated the effectiveness compared with word-mangling rule-based methods. Ma, et al. [24] leveraged a Markov model to learn the distribution of passwords. They showed that their proposed method achieved slightly better performance than PCFGs in password cracking when normalization and smoothing were used. Melicher, et al. [26] designed a password model using a recurrent neural network [34], which achieves improved accuracy in password strength measurement. Pasquini, et al. [30] utilized Generative Adversarial Networks (GAN) [18] to train a password generative model. They showed that the trained model can be used to produce passwords more effectively if a password template is known, due to the strong locality in the latent space of the generative model. Most recently, Xu, et al. [57] improved password guessing by learning a bi-direction transformer model.

Recent works showed that password guessing can be improved by utilizing account holder PII and passwords used by the same user at other sites. Wang, et al. [47] proposed a PCFG model named *TarGuess* where PII is considered in the model training. Pal, et al. [29] studied the case that attacker utilized the passwords used by the same users leaked from *another* site to crack password, known as *credential tweaking*. They trained a *Pass2Path* model by a recurrent neural network to simulate credential tweaking, which compromised at least 16% of user accounts in their tests. He, et al. [19] considered a similar threat model but improved the compromising rate using a deep neural transformer [42]. Recently, Wang, et al. [49] modeled password reuse behavior by a multi-step generative model, which improved the password guessing. In this paper, we adapted some of these techniques from *credential tweaking* (Tweak and P2P as described in Sec. 3) for honeyword generation.

2.3 Honeyword-Based Systems

Our study is agnostic to system designs leveraging honeywords, whether they be symmetric or asymmetric. Asymmetric designs are ones that detect honeyword entry using a secret that the attacker is presumed to be unable to capture in the breach. For example, the original honeyword-system design [22] leverages a trusted server called a *honeychecker* that holds the index of the legitimate password for each account, which the login server consults to determine whether a login attempt uses a honeyword or the legitimate password. This honeychecker is assumed to keep its indices secret despite the login server's breach. Other asymmetric designs include ErsatzPasswords [2] and Lethe [12].

By contrast, a symmetric design is one where the attacker is allowed to capture all state used for honeyword-entry detec-

Experiment $\mathsf{Expt}^{\mathrm{FPP}}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A})$
$(p,X) \leftarrow \mathcal{U}()$
$H \leftarrow \mathcal{H}_n(p)$
$G \leftarrow \mathcal{A}(p)$
if $|G| \leq \beta \wedge |G \cap H| \geq \alpha$
then return 1
else return 0

$$\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A}) \stackrel{\text{def}}{=} \mathbb{P}\left(\mathsf{Expt}^{\mathrm{FPP}}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A}) = 1\right)$$

$$\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta} \stackrel{\text{def}}{=} \max_{\mathcal{A}} \mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A})$$

(a) False-positive probability

Experiment $\mathsf{Expt}^{\mathrm{FNP}}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B})$
$(p,X) \leftarrow \mathcal{U}()$
$H \leftarrow \mathcal{H}_n(p)$
$G \leftarrow \mathcal{B}(H \cup \{p\}, X)$
if $|G \cap H| < \alpha \wedge p \in G$
then return 1
else return 0

$$\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B}) \stackrel{\text{def}}{=} \mathbb{P}\left(\mathsf{Expt}^{\mathrm{FNP}}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B}) = 1\right)$$

$$\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha} \stackrel{\text{def}}{=} \max_{\mathcal{B}} \mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B})$$

(b) False-negative probability

Figure 1: Measures for breach detection by honeywords. A false-positive attacker $\mathcal{A}$ tries to trick an unbreached site into detecting that it has been breached (Fig. 1a). A false-negative attacker $\mathcal{B}$ attempts to access an account after breaching the site, without alerting the site to its breach (Fig. 1b).

tion when he breaches the site. An example of a symmetric design is Amnesia [52]. In this design, the attacker captures all the information needed to undetectably access an account—possibly using a honeyword—at a site it breaches. However, the act of doing so configures the site to learn of its breach once the legitimate user accesses the site subsequently, using a different password. That is, in Amnesia, the use of two different passwords to enter an account is what alerts the site to its breach, since one must be a honeyword.

3 Background

3.1 Definitions

Honeywords are decoy passwords added to each account entry in a credential database. The principle behind honeywords is that since the legitimate user does not know the honeywords generated for her account, the only party who is able to enter those honeywords is an attacker who discovered them by breaching the credential database. As such, login attempts using honeywords should be taken as compelling evidence of a database breach.

To make this principle precise, we define the false-positive and false-negative probabilities of a honeyword scheme in a way that abstracts away the details of the system leveraging them. We do so using the experiments shown in Fig. 1 and described in text below. In these experiments, a random user is modeled by a randomized algorithm $\mathcal{U}$, by which the user selects her password $p \in \{0,1\}^*$ for a site. The invocation $\mathcal{U}()$ outputs not only the password p, but also auxiliary information $X \subset \{0,1\}^*$ that is correlated with p and that the attacker might learn. In this work, X will be passwords set by the same user at other sites, though other works have considered other types of auxiliary information (e.g., [46]). Given p, the site selects honeywords for this account using the randomized algorithm $\mathcal{H}_n$, which outputs a set H where $|H| = n$ and $p \notin H$.

A *false-positive attacker* $\mathcal{A}$ attempts to trigger a breach alarm at this site even though it has not breached the site, by leveraging its knowledge of p and $\mathcal{H}_n$ to guess honeywords in H. In this work, we consider the worst case where $\mathcal{A}$ is permitted to know p since $\mathcal{A}$ might represent a legitimate user of this site or because it might represent an outsider who, say, phished p. $\mathcal{A}$ might know X but X does not help in guessing H if p is already known. $\mathcal{A}$ is provided knowledge of the honeyword-generation algorithm $\mathcal{H}_n$ to provide a conservative analysis.[1] $\mathcal{A}$'s probability of triggering an alarm is defined in Fig. 1a, where $\alpha \geq 1$ is the number of honeywords whose entry will trigger a breach alarm and where $\beta \geq 1$ denotes the number of login attempts $\mathcal{A}$ is permitted to attempt for this account. In words, given p (along with $\mathcal{U}$, $\mathcal{H}_n$, α, and β, which are public parameters of the experiment), $\mathcal{A}$ wins by outputting a set G that it can enter in its budget of login attempts ($|G| \leq \beta$) and that will trigger an alarm ($|G \cap H| \geq \alpha$). Traditionally the threshold for raising a breach alarm has typically been set to $\alpha = 1$, though this definition permits other values; A larger α implies a more stringent condition for raising an alarm. $\mathcal{A}$'s *false-positive probability* $\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A})$ is the probability that $\mathcal{A}$ wins, and the overall false-positive probability $\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}$ is $\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A})$ for the attacker algorithm $\mathcal{A}$ that maximizes that probability. When the parameters $\mathcal{U}$, $\mathcal{H}_n$, α, and β are clear from context, we will abbreviate $\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}(\mathcal{A})$ and $\mathsf{FPP}_{\mathcal{U},\mathcal{H}_n,\alpha,\beta}$ as $\mathsf{FPP}(\mathcal{A})$ and FPP, respectively, to simplify notation.

In contrast, a *false-negative attacker* $\mathcal{B}$ is an attacker who attempts to access this user's account after breaching the site but without alerting the site that it has been breached. This adversary's advantage in doing so is defined in Fig. 1b. In words, $\mathcal{B}$ obtains the set $H \cup \{p\}$, sometimes called the *sweetwords* for this account, as well as auxiliary information X. The set $H \cup \{p\}$ is sweetwords are recovered by the attacker from the salted hash file. $\mathcal{B}$ then wins if it outputs a set G that will not trigger an alarm ($|G \cap H| < \alpha$) and that permits it to access the account ($p \in G$). Here we presume that $G \subseteq H \cup \{p\}$, since passwords other than the sweetwords outside $H \cup \{p\}$ offer no help for $\mathcal{B}$ to achieve his goals. Consequently, we drop β as a parameter of the experiment; since $\beta \geq \alpha \geq |G|$, it does not constrain $\mathcal{B}$'s choice of G. Again, traditionally the threshold for raising a breach alarm has been set

[1] Allowing $\mathcal{A}$ knowledge of $\mathcal{H}_n$ conforms with general security design principles; e.g., "Do not rely on secret designs, attacker ignorance, or *security by obscurity*." [41, p. 21]. In our context specifically, if the attacker knows only that $\mathcal{H}_n$ is one of several alternatives, it can try each alternative via a different account. In this sense, our measure is analogous to the *min auto* approach to measuring password strength [28, 40], in which the strength of a password is measured by the number of tries to guess it, under the guessing strategy (from among several) that minimizes that number.

to $\alpha = 1$, in which case the probability with which $\mathcal{B}$ guesses p from the sweetwords $H \cup \{p\}$ on the first try (i.e., $|G| = 1$) is called the *flatness* of the honeyword scheme. $\mathcal{B}$'s *false-negative probability* $\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B})$ is the probability that $\mathcal{B}$ wins, and the overall false-negative probability $\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}$ is $\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B})$ for the attacker $\mathcal{B}$ that maximizes that probability. When the parameters $\mathcal{U}$, $\mathcal{H}_n$, and α are clear from context, we will abbreviate $\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}(\mathcal{B})$ and $\mathsf{FNP}_{\mathcal{U},\mathcal{H}_n,\alpha}$ as $\mathsf{FNP}(\mathcal{B})$ and FNP, respectively, to simplify notation.

A honeyword-generation algorithm $\mathcal{H}_n$ can at best achieve $\mathsf{FPP} \approx 0$ and $\mathsf{FNP} = \frac{\alpha}{n+1}$. Our research evaluates the extent to which known honeyword-generation algorithms, described in Sec. 3.2, approach this ideal. When considering false-negative attackers, we will evaluate an attacker who prioritizes accounts by its perceived likelihood of success in guessing the account password p, by refining $\mathcal{U}$ to represent "easy" users for whom $(H \cup \{p\}) \cap X \neq \emptyset$, likely due to exact password reuse across accounts; "medium" users who are not "easy" but for whom there are elements of $H \cup \{p\}$ and X that are close to one another (in a sense we will define later), likely because the user set passwords at her other accounts that are similar to p (partial password reuse); or "hard" users for whom neither condition holds.

3.2 Honeyword-Generation Algorithms

In this section, we introduce honeyword-generation algorithms, some of which have been introduced in previous works [13, 22, 48]. Generally, honeyword-generation algorithms can be classified into two groups: *password-independent* algorithms and *password-dependent* algorithms. We included more details of these algorithms in the full paper [20, App. C].

3.2.1 Password-Independent Honeyword Generation

Password-independent algorithms generate honeywords independently of the account passwords. They do so by sampling password candidates from *password models* pretrained on a multiset of passwords. In this work, we consider four widely used password models: list model [46], probabilistic context-free grammar model [55], Markov model [24], and recurrent neural network [26], and their combination [48]. We denote these generation methods as List, PCFG, Markov, RNN, and Combo, respectively.

3.2.2 Password-Dependent Honeyword Generation

Password-dependent algorithms generate honeywords that are dependent on the account passwords. These algorithms include password-strength-dependent methods and password-context-dependent methods.

Password-strength-dependent methods generate honeywords whose strength is equal or similar to the input password p. These methods still leverage password models such as List, PCFG, Markov, RNN, or their combination but select a sampled candidate as a honeyword if and only if its strength is equal to that of the input password. However, if the input password is weak, it might be difficult to generate n honeywords with equal password strength, under the hypothesis that user-chosen passwords follow a Zipf distribution (e.g., [45]). So, in this work, we relax this requirement so that a sampled candidate will be used as a honeyword if its length equals the length of the input password. We denote this algorithm for generating honeywords from List, PCFG, Markov, RNN, or a combined method by List∗, PCFG∗, Markov∗, RNN∗, and Combo∗.

Password-context-dependent methods generate honeywords by modifying the input password. Here we consider four types of techniques: targeted password model-based generation, LLM-based generation, random replacement-based tweaking, and DNN-based tweaking.

Targeted password model-based generation: These methods generate honeywords from password models that learn a distribution of *password templates* [48]. Here a password template is a pattern describing passwords set by the same user at different sites, wherein common substrings are indicated in the template using a special tag `pwd_str`. For example, the template "`pwd_str` z" might be generated from "bike123z" and "bike123" if these passwords were set by the same user at two different sites. Password models like PCFG are pretrained on a multiset of password templates, as targeted password models. Then, honeywords are generated by sampling templates from the targeted password models and replacing `pwd_str` in the templates with the input password. We denote these generation methods from List, PCFG, Markov, RNN, or a combined method by List#, PCFG#, Markov#, RNN#, and Combo#.

LLM-based generation: These techniques generate honeywords by querying a large language model like GPT-3 [4] with prompts based on the input password. We consider a recently proposed method, chunk-level GPT-3 (CGPT3) [58].

Random replacement-based tweaking: These techniques generate honeywords by randomly changing some characters of the input password or similar passwords. We consider chaffing-by-tweaking or CBTt [22], which generates honeywords by randomly replacing the last t characters of the input password with characters of the same type; CBT∗ [13], which generates honeywords by similarly replacing all the characters; and chaffing-by-a-hybrid-model (CHM [13]).

DNN-based tweaking: DNN-based tweaking techniques leverage DNNs to tweak the chosen password to generate its honeywords. We consider a deep tweak model (Tweak) [19] and tweaking path model (P2P) [29], which are adapted from similar constructions originally developed to crack passwords [19,29]. The deep tweak model is a DNN that, on input a password, outputs a tweaked password. The tweaking path model inputs a password and outputs an edit path that is used to change the input password.

4 User-Chosen Passwords

The first case we consider is when $\mathcal{U}$ is an algorithm implemented by an average human user, and X is a multiset of passwords chosen by the same user at other sites. In this case, we show that the field has yet to identify *any* honeyword-generation algorithm that achieves small FNP and FPP simultaneously. Intuitively, this is true because when a user selects passwords without automated help (i.e., $\mathcal{U}$ is an average user), then an attacker who guesses passwords G that are similar to passwords in X will be highly effective in either inducing false detections (a high $\mathsf{FPP}(\mathcal{A})$) or avoiding true detection (a high $\mathsf{FNP}(\mathcal{B})$). On the one hand, if $\mathcal{H}_n(p)$ outputs honeywords dissimilar to p, then since users often choose p similar to elements of X, it will be relatively easy for an attacker $\mathcal{B}$ to select p from $H \cup \{p\}$ as the one most similar to passwords in X. So, for $\mathsf{FNP}(\mathcal{B})$ to be small, $\mathcal{H}_n(p)$ must output at least some honeywords that are similar to p. On the other hand, the more it does so, the easier it is for an attacker $\mathcal{A}$ to induce false detections by guessing passwords G similar to passwords in X.

4.1 Attack Strategies

In this section, we introduce the false-positive attacker $\mathcal{A}$ and the false-negative attacker $\mathcal{B}$ that we use in the evaluation of $\mathsf{FPP}(\mathcal{A})$ and $\mathsf{FNP}(\mathcal{B})$, respectively.

False-positive attacker $\mathcal{A}$: In the evaluation of $\mathsf{FPP}(\mathcal{A})$, recall that $\mathcal{A}$ is given access to p. The attacker $\mathcal{A}$ leverages the honeyword-generation algorithm $\mathcal{H}$ on input p to generate a set of honeyword candidates. Then, if applicable, it sorts the candidates by the probabilities assigned by the honeyword-generation algorithm and uses the top β candidates as the guessed honeywords G; otherwise, it picks β candidates uniformly at random without replacement as G.

False-negative attacker $\mathcal{B}$: We evaluate $\mathsf{FNP}(\mathcal{B})$ for user-chosen passwords as follows. Given passwords X, $\mathcal{B}$ leverages a metric function $d(\cdot) : \{0,1\}^* \times \{0,1\}^* \rightarrow \mathbb{R}$ to measure the similarity between the elements of X and the sweetwords $H \cup \{p\}$, and ranks each sweetword based on its similarity to the most similar element of X. The top α ranked sweetwords are used to guess p.

4.2 Model to Measure Password Similarity

In the evaluation of $\mathsf{FNP}(\mathcal{B})$, we need to define a metric function that inputs a pair of passwords and returns a score reflecting the similarity between the inputs. To formulate such a metric function, we designed a similarity model $f(\cdot) : \{0,1\}^* \rightarrow \mathbb{R}^d$ by a deep neural network, which takes as input a password p and outputs its *latent representation* such that the cosine similarity between any two latent representations $f(p)$ and $f(p')$ grows with the probability that $\mathcal{U}()$ would have output both, i.e., with $\mathbb{P}\big(p' \in X \mid (p,X) \leftarrow \mathcal{U}()\big)$.

The similarity model is used to learn the embedding of passwords. Learning such an embedding of passwords into a latent space is essentially a *metric learning* problem [37, 50]. Therefore, we applied contrastive learning, which is one of the most widely used frameworks to train a model to perform this embedding so as to maximizing cosine similarity between positive (similar) pairs while minimizing cosine similarity of negative (dissimilar) pairs [7]. Training a contrastive model is performed in *batches*, each a multiset $B \subseteq \{0,1\}^* \times \{0,1\}^*$. Each $(p, p') \in B$ consists of similar passwords (intuitively, for which $\mathbb{P}\big(p' \in X \mid (p,X) \leftarrow \mathcal{U}()\big)$ is high), whereas for any $(p'', p''') \in B \setminus \{(p,p')\}$, p and p'' are presumed to be dissimilar, as are p' and p'''. Training for a contrastive learning model of password similarity, therefore, updates f to minimize a *loss function*, which typically would take the form

$$\operatorname*{avg}_{(p,p')\in B} -\log \frac{\exp(\mathsf{sim}(f(p),f(p')))}{\sum\limits_{\substack{(p'',p''')\in B:\\ (p'',p''')\neq(p,p')}} \begin{pmatrix}\exp(\mathsf{sim}(f(p),f(p''))) + \\ \exp(\mathsf{sim}(f(p'),f(p''')))\end{pmatrix}} \qquad (1)$$

where sim denotes cosine similarity (see Chen, et al. [7]). Such updates with all the data samples from the training dataset passed through the trained model constitute one *epoch*. The design and training of the similarity model are described in the full paper [20, App. B].

4.3 Evaluation

In this subsection, we detail our evaluation of the user-chosen password case, which includes the used dataset, and the experimental results for $\mathsf{FPP}(\mathcal{A})$ and $\mathsf{FNP}(\mathcal{B})$.

4.3.1 The Dataset

The dataset we used in the case of user-chosen passwords is the *4iQ* dataset [5], consisting of 1.4 billion (email, password) pairs, of which 1.1 billion emails and 463 million passwords are unique. Others attribute the *4iQ* dataset to various leaks from LinkedIn, Myspace, Badoo, Yahoo, Twitter, Zoosk, and Neopet, and have used it to analyze users' choices of passwords across sites [29] (despite the possibility of some being automatically generated). Our use of leaked passwords was approved by our IRB, which specified protections in our handling of this data

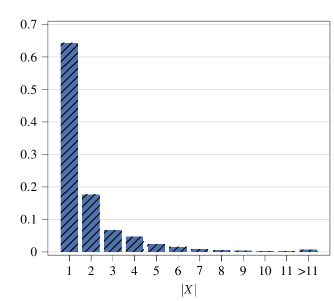

Figure 2: Distribution of $|X|$

Statistic	Value
Total number of users	195,894,983
Total number of passwords	563,743,130
Average passwords per user	2.877
Average distinct passwords per user	1.961
Percentage of users reusing passwords	48.507%

Table 1: Statistics of the preprocessed dataset

(who could access the data, what results could be reported, etc.). In order to use *4iQ*, we preprocessed the dataset by referring to previous works (e.g., [29]).

- **Cleaning:** We removed any (email, password) pairs that satisfied any of the following conditions: the password contained non-ASCII characters, the space character, or a substring of 20 (or more) hex characters; the password had a length of less than 4 or more than 30; or the email contained non-ASCII characters or the space character.
- **Joining by email and username:** For each email address addr appearing in the dataset, we collected the passwords appearing with that email address into a multiset S_{addr}. Then we merged some password multisets S_{addr} as follows: two multisets were merged if they contained at least one password in common and if the username parts of their email addresses were the same. We then eliminated each S_{addr} containing only one password or $> 1{,}000$ passwords. In the resulting dataset, around 48% of users reused passwords, which is within the range between 43% and 51% estimated by previous work (e.g. [9]). More statistics about the resulting dataset are shown in Table 1.
- **Splitting into training and testing sets:** Of the 195,894,983 password multisets that remained, 80% (156,722,455 multisets with 451,020,019 passwords) were set aside as training sets $D_{\mathsf{u}}^{\mathsf{tr}}$ used to train models. The other 20% (39,172,528 multisets with 112,723,111 passwords) of the password multisets were set aside as testing sets $D_{\mathsf{u}}^{\mathsf{te}}$. When evaluating $\mathsf{FNP}(\mathcal{B})$ and $\mathsf{FPP}(\mathcal{A})$, the algorithm $\mathcal{U}$ was implemented by choosing p and the members of X without replacement from a single multiset S_{addr} chosen uniformly at random from the testing sets, and returning (p,X) as the result with $X = S_{\mathsf{addr}} \setminus \{p\}$. $|X|$ represents the amount of attacker's knowledge about this user's passwords at other sites. Its distribution in $D_{\mathsf{u}}^{\mathsf{te}}$ is shown in Fig. 2.

4.3.2 Experimental Results

We now report $\mathsf{FPP}(\mathcal{A})$ and $\mathsf{FNP}(\mathcal{B})$ for the attackers $\mathcal{A}$ and $\mathcal{B}$ described in Sec. 4.1. To depict the tradeoffs between these measurements, we plot them against one another as α is varied. When evaluating $\mathsf{FNP}(\mathcal{B})$, we isolate three subcases, to permit modeling of an attacker who prioritizes accounts based on similarities between $H \cup \{p\}$ and X per account. We measured such similarity based on definitions like those for password reuse introduced in previous work (e.g., [31]). Specifically, "easy" accounts are those for which $(H \cup \{p\}) \cap X \neq \emptyset$; "medium" accounts are those for which $(H \cup \{p\}) \cap X = \emptyset$ but there is a sweetword in $H \cup \{p\}$ that shares a substring of length at least four characters with some password in X; and "hard" accounts are those that are neither "easy" nor "medium". The percentages of accounts of different hardness are shown in Table 2.

	$n=19$			$n=99$		
$\mathcal{H}_n$	easy	med	hard	easy	med	hard
List	43.37	16.00	40.63	43.56	19.62	36.82
Markov	43.36	15.96	40.68	43.57	19.78	36.65
PCFG	43.36	15.59	41.05	43.47	18.40	38.13
RNN	43.36	16.01	40.63	43.59	19.57	36.84
Combo	43.33	15.98	40.69	43.55	19.27	37.18
List∗	43.37	16.07	40.56	43.41	19.39	37.20
Markov∗	43.33	16.05	40.62	43.38	19.57	37.05
PCFG∗	43.35	15.66	40.99	43.38	18.68	37.94
RNN∗	43.34	15.92	40.74	43.40	19.46	37.14
Combo∗	43.37	15.94	40.69	43.38	19.70	36.92
List#	44.11	19.25	36.64	43.55	15.80	40.65
Markov#	44.00	18.91	37.09	43.47	15.92	40.61
PCFG#	43.47	15.80	40.73	44.00	19.13	36.87
RNN#	43.54	15.64	40.82	43.98	19.11	36.91
Combo#	43.49	15.74	40.77	44.09	18.65	37.26
CGPT3	44.54	14.16	41.28	44.98	14.28	40.72
CBT3	43.35	14.87	41.78	43.34	15.28	41.38
CBT4	43.33	14.97	41.73	43.33	15.31	41.36
CBT∗	43.53	14.92	41.55	43.84	15.56	40.60
CHM	43.46	15.33	41.21	43.91	15.83	40.26
Tweak	44.80	14.22	40.98	45.89	14.67	39.44
P2P	46.05	12.36	41.59	47.39	12.18	40.43

Table 2: Percentages of accounts of different hardness for a false-negative attacker $\mathcal{B}$, discussed in Sec. 4.3.2

Fig. 3 shows the tradeoffs between $\mathsf{FPP}(\mathcal{A})$ and $\mathsf{FNP}(\mathcal{B})$ for $n = 19$ honeywords and $\beta = 1000$, for the various honeyword-generation algorithms described in Sec. 3. RNN and its variants achieved similar performance to List, PCFG, Markov, Combo, and their variants, and thus we only show the results from List, PCFG, Markov, Combo, and their variants in Fig. 3; results for RNN and its variants are in the full paper [20, App. C]. In each plot, there are four curves presenting the overall tradeoff ("all") and those of three subcases: "easy", "medium", and "hard". In each curve, markers highlight the $\mathsf{FPP}(\mathcal{A})$ vs. $\mathsf{FNP}(\mathcal{B})$ tradeoff at a specific values of α ranging from $\alpha = 1$ to n. Intuitively, a smaller α yields lower $\mathsf{FNP}(\mathcal{B})$ but higher $\mathsf{FPP}(\mathcal{A})$ and so a marker closer to the top left corner. Increasing α to n yields a higher $\mathsf{FNP}(\mathcal{B})$ but lower $\mathsf{FPP}(\mathcal{A})$ and so a marker closer to the bottom right corner. We stress that $\beta = 1000$ yields an optimistic evaluation of $\mathsf{FPP}(\mathcal{A})$. For example, Florêncio, et al. [16] recommend that an account should withstand targeted online password-guessing attacks of 10^6 attempts in practice. As such, arguably

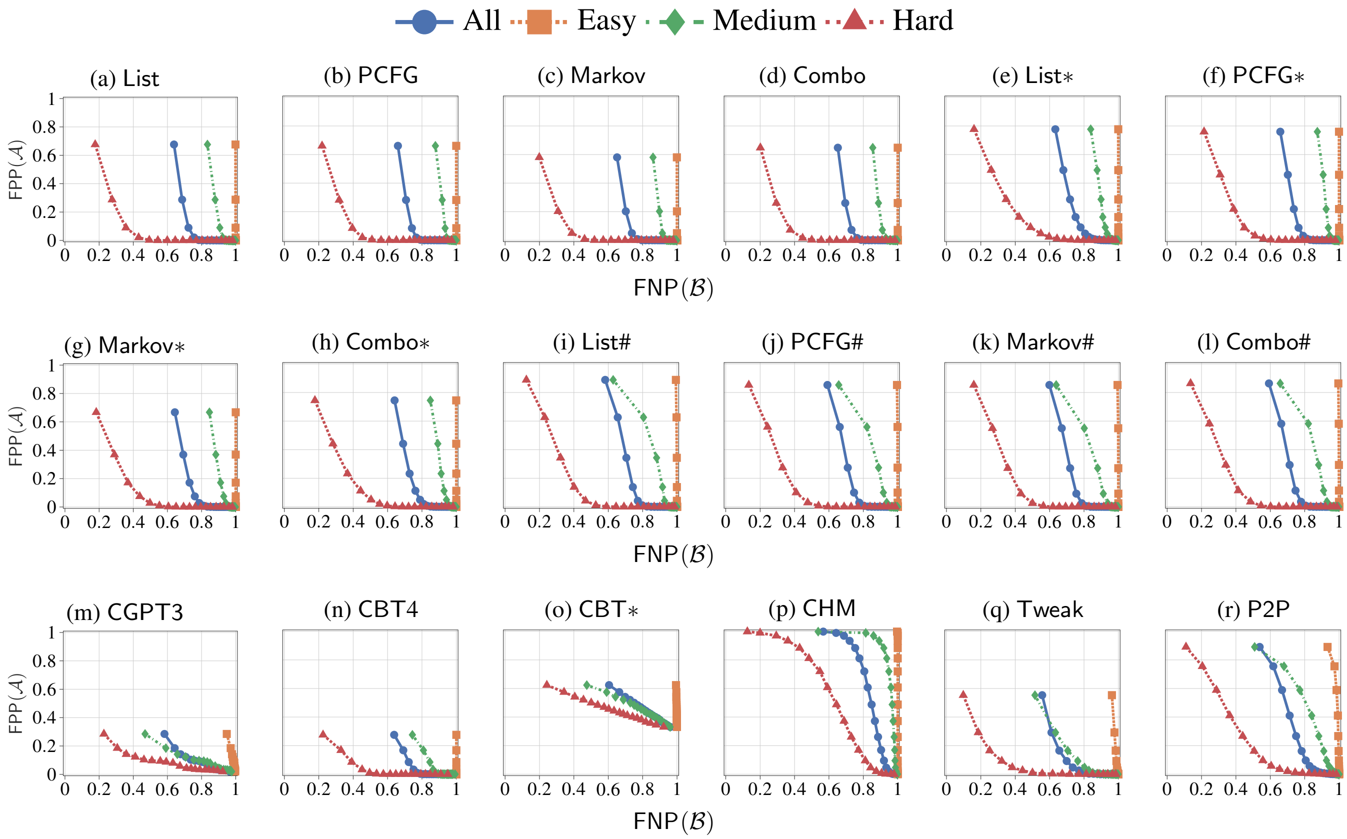

Figure 3: $\mathsf{FPP}(\mathcal{A})$ vs. $\mathsf{FNP}(\mathcal{B})$ as α is varied, for the case of user-chosen passwords ($n = 19$, $\beta = 1000$). The best $\mathsf{FNP}(\mathcal{B})$ are 0.54 (P2P, Fig. 3r), 0.56 (Tweak, Fig. 3q), 0.57 (CHM, Fig. 3p), and 0.58 (CGPT3, Fig. 3m); all others have $\mathsf{FNP}(\mathcal{B}) > 0.59$. All suffer $\mathsf{FPP}(\mathcal{A}) > 0.27$ at $\alpha = 1$. Those that reach $\mathsf{FPP}(\mathcal{A}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.81$.

$\beta = 1000$ is $1000\times$ too small.

An ideal honeyword-generation algorithm would achieve $\mathsf{FPP}(\mathcal{A}) \approx 0$ and $\mathsf{FNP}(\mathcal{B}) = \frac{1}{n+1}$ (which is 0.05 when $n = 19$) at $\alpha = 1$. Unfortunately, no known honeyword algorithm comes close. As seen in Fig. 3, the best $\mathsf{FNP}(\mathcal{B})$ that the honeyword-generation techniques accomplish overall is 0.54 (P2P, Fig. 3r), 0.56 (Tweak, Fig. 3q), 0.57 (CHM, Fig. 3p), and 0.58 (CGPT3, Fig. 3m); all others have $\mathsf{FNP}(\mathcal{B}) > 0.59$. When we consider the attacker prioritizing "easy" accounts, $\mathsf{FNP}(\mathcal{B})$ of P2P, CGPT3, and Tweak are at least 0.93, 0.95, and 0.96, respectively, while others have $\mathsf{FNP}(\mathcal{B}) \approx 1$. This indicates that the false-negative attacker can break at least 43% accounts by targeting the "easy" ones, with only P2P, CGPT3, and Tweak presenting any significant chance of catching the attacker. That said, when such an attacker wants to guess more account passwords, i.e., targeting the "medium" accounts after the "easy" ones, the probability of inducing an alarm will increase with number of attacked accounts since $\mathsf{FNP}(\mathcal{B}) < 0.88$ for the "medium" subcase when $\alpha = 1$. The four most successful algorithms (P2P, Tweak, CHM, and CGPT3) are password-context-dependent techniques that generate honeywords similar to the account password, and thus it is more challenging for $\mathcal{B}$ to distinguish the account password from honeywords produced by these algorithms than from those of the other methods. We conclude that honeywords more similar to the account password yield a lower $\mathsf{FNP}(\mathcal{B})$, though one that is still far from $\frac{1}{n+1}$ due to password reuse.

However, P2P has $\mathsf{FPP}(\mathcal{A}) \approx 0.89$ at $\alpha = 1$, where most others have lower $\mathsf{FPP}(\mathcal{A})$. The only exception is CHM, which includes a deterministic step that searches for nearest neighbors of the account password and thus yields a high false-positive rate, $\mathsf{FPP}(\mathcal{A}) \approx 1$. While P2P is the best technique for generating honeywords similar to the account password, it is almost the easiest for the false-positive attacker to guess the generated honeywords with p known. Still, no generation method achieves $\mathsf{FPP}(\mathcal{A}) \leq 0.27$ at $\alpha = 1$. Growing α of course reduces $\mathsf{FPP}(\mathcal{A})$ but increases $\mathsf{FNP}(\mathcal{B})$: all methods capable of reaching $\mathsf{FPP}(\mathcal{A}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.81$ overall, $\mathsf{FNP}(\mathcal{B}) \approx 1$ for the "easy" subcase, and $\mathsf{FNP}(\mathcal{B}) > 0.91$ for the "medium" subcase.

A natural method to decrease $\mathsf{FNP}(\mathcal{B})$ would be to increase the number n of honeywords, but the more pronounced effect of doing so is increasing $\mathsf{FPP}(\mathcal{A})$, instead. Indeed, Fig. 4 shows the impact of increasing n to $n = 99$. As seen there, an

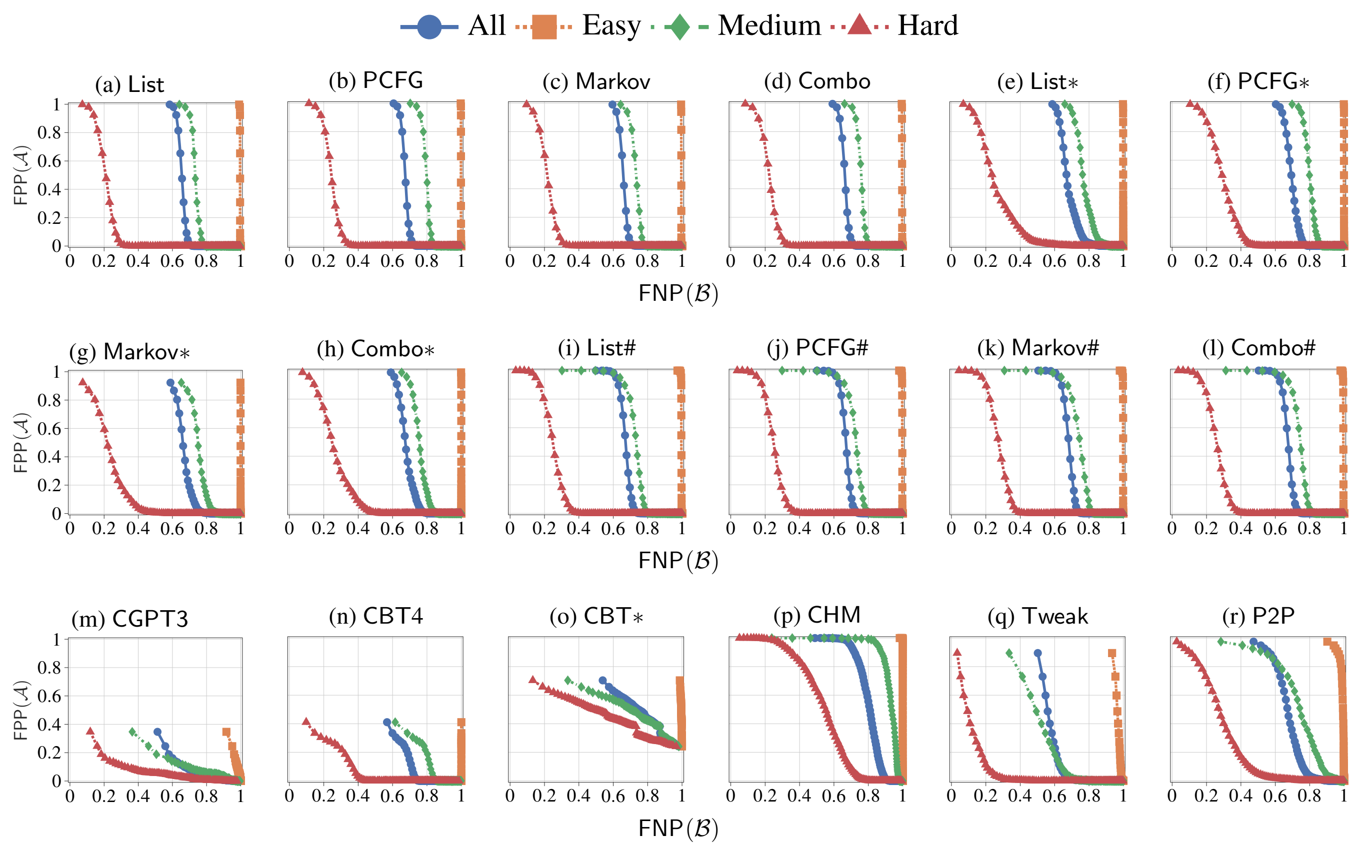

Figure 4: $\mathsf{FPP}(\mathcal{A})$ vs. $\mathsf{FNP}(\mathcal{B})$ as α is varied, for the case of user-chosen passwords ($n = 99$, $\beta = 1000$). The best $\mathsf{FNP}(\mathcal{B})$ are 0.47 (P2P, Fig. 4r), 0.49 (CHM, Fig. 4p), and 0.50 (Tweak, Fig. 4q); all others have $\mathsf{FNP}(\mathcal{B}) > 0.50$. All suffer $\mathsf{FPP}(\mathcal{A}) > 0.34$ at $\alpha = 1$. Those that reach $\mathsf{FPP}(\mathcal{A}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.72$.

order-of-magnitude increase in n resulted in a slight improvement to $\mathsf{FNP}(\mathcal{B})$ in each case, but a more substantial increase to $\mathsf{FPP}(\mathcal{A})$.

To summarize, honeyword-generation techniques like Combo# that have been demonstrated to have good flatness in previous works (e.g., [48]) fail to achieve a low false-negative rate in our threat model, particularly not at settings of α to ensure a small false-positive rate. Among the honeyword-generation techniques we consider, P2P achieves the best FNP but has a high FPP. Most other methods have lower FPP but a higher FNP. Regardless, in the case of user-chosen passwords, no existing algorithm achieves low rates of both false positives and false negatives. In addition, when the attacker targets the "easy" accounts that are approximately 43% of users, all the honeyword-generation methods are ineffective in detecting a breach at settings of α achieving $\mathsf{FPP}(\mathcal{A}) \approx 0$.

5 Algorithmically Generated Passwords

The second case we consider is when $\mathcal{U}$ is implemented using a password-generating algorithm. To our knowledge, this case has not been considered in prior honeyword research, and with good reason: Under the best practice of storing only preimage-resistant hashes of passwords, it should be exceptionally difficult for an attacker who breaches a site's database to recover algorithmically generated passwords, due to their comparatively high strength. For this reason, algorithmically generated passwords in a breached credential database are primarily at risk if the hash function is less preimage-resistant than initially thought or when the site's database was reversibly encrypted—which, while not best practice, is necessary in some use cases (e.g., [27])—and the false-negative attacker recovered the decryption key along with the database.

We assume there is a large but limited number Y of password generators denoted as $\{\mathcal{U}_y\}_{y=1}^{Y}$, each of which is defined by an algorithm and values of user-configurable parameters. We assume that each user determines $\mathcal{U}$ by choosing a generator uniformly at random from $\{\mathcal{U}_y\}_{y=1}^{Y}$, and that each user stays with its choice. To justify this assumption, in Sec. 6.4 we report a brief study we did using the password policies of 20 commonly visited websites and Tranco Top 1M websites [32], where we found that setting passwords at these websites in a random order would permit the user to retain her chosen password-generation configuration for > 6.3 sites in expectation, before encountering a site for which the user's configuration was inconsistent. This finding is consistent with Alroomi et al. [3], who reported that only 15% sites have character constraints on password creation.

We assume the length of the generated passwords is one parameter that users can configure. Some password managers permit user configuration of allowable symbols, as well. Similarly, password managers that enable generation of easy-to-read passwords might avoid use of certain characters that are ambiguous in some fonts (e.g., "1" vs. "l" in sans-serif fonts). Password managers that generate easy-to-say passwords might restrict the symbols used in different positions of a password. We will see examples below. The user's choice of these parameters will generally be unknown to the defender, except as revealed by the account password p.

In this section, we analyze the contribution of honeywords for detecting credential database breaches for accounts with algorithmically generated passwords. In the full paper [20, App. E], we show that honeyword-generation methods used in the user-chosen password case fail to achieve both low false-negative rate and low false-positive rate for algorithmically generated passwords. Although utilizing password-generation algorithms to generate honeywords can do better, in this section we show that the choice of selected generator is critical to achieving a low false-negative rate.

5.1 Attack Strategies

In this section, we introduce the false-positive attacker $\mathcal{A}$ and the false-negative attacker $\mathcal{B}$ used in the evaluation of $\mathsf{FPP}(\mathcal{A})$ and $\mathsf{FNP}(\mathcal{B})$, respectively, when account passwords are generated algorithmically.

False-positive attacker $\mathcal{A}$: $\mathcal{A}$ uses the same strategy used in the case of user-chosen passwords in Sec. 4.1. Specifically, the attacker $\mathcal{A}$ leverages the honeyword-generation algorithm $\mathcal{H}$ to generate a set of candidates and sorts the candidates by their assigned probabilities, if applicable. Finally, it picks the top β candidates as the guessed honeywords G.

False-negative attacker $\mathcal{B}$: $\mathcal{B}$ was implemented as follows. Given X, $\mathcal{B}$ leverages a classifier $f(\cdot) : \{0,1\}^* \to [0,1]^Y$ that outputs a confidence score per possible class. The construction of this classifier is described in the full paper [20, App. D]. $\mathcal{B}$ classifies each element of X using f, using the highest-scored generator for each $p' \in X$ as a "vote" for the password generator that the user employs; the password generator obtaining the most such votes is denoted $\mathcal{U}_{y_\mathcal{B}}$. Then $\mathcal{B}$ assigns scores to the sweetwords from $H \cup \{p\}$ as follows: if the length of the sweetword is the same as those in X, $\mathcal{B}$ utilizes the classifier $f(\cdot)$ to value the sweetword by the confidence score of being from class $y_\mathcal{B}$; otherwise, $\mathcal{B}$ will value it by 0. The attacker ranks the sweetwords based on the assigned scores and uses the top α sweetwords as G.

5.2 Generating Honeywords Using Algorithmic Password Generators

The honeyword-generation methods introduced in Sec. 3.2 do not fare well (in terms of false-negative probability) when the account password is generated algorithmically. Intuitively, the password-independent honeyword generators fail to achieve a low $\mathsf{FNP}(\mathcal{B})$ since the honeywords they generate are user-chosen passwords, which makes it easy for $\mathcal{B}$ to distinguish the algorithmically generated account password from the honeywords. Many password-dependent generators do little better, because even though the account password is algorithmically generated, these models are trained on artifacts of human behavior, which renders the honeywords recognizable to $\mathcal{B}$. The primary exceptions are CBT3 and CBT4, which are not trained at all. These can achieve a low $\mathsf{FNP}(\mathcal{B})$, though still with a too-high $\mathsf{FPP}(\mathcal{A})$. We have empirically demonstrated these findings in the full paper [20, App. E].

Therefore, here we consider the use of algorithmic password generators to generate honeywords for algorithmically generated passwords submitted by the user. Given an account password p, the honeyword system selects a generator from $\{\mathcal{U}_y\}_{y=1}^Y$ and then leverages the selected generator to generate n honeywords. We categorize the methods based on the selection strategy, as follows:

- FXED: Given a fixed $\mathcal{U}_{y_{\mathsf{fx}}} \in \{\mathcal{U}_y\}_{y=1}^Y$, $\mathcal{H}_n$ samples n distinct honeywords using $\mathcal{U}_{y_{\mathsf{fx}}}$ to build H.
- RAND: $\mathcal{H}_n$ samples a $\mathcal{U}_y$ uniformly from $\{\mathcal{U}_y\}_{y=1}^Y$ and builds H by sampling n distinct honeywords using $\mathcal{U}_y$.
- CLSD: $\mathcal{H}_n$ classifies the account password into one of Y classes, indicating the generator $\mathcal{U}_y$ most likely to have generated it. $\mathcal{H}_n$ then builds H by sampling n distinct honeywords using $\mathcal{U}_y$.

5.3 Evaluation

5.3.1 Dataset

The datasets we used to evaluate honeyword-generation strategies in the case of algorithmically generated passwords were synthetically produced by querying online password generators.[2] Specifically, after browser-integrated password managers (Google Password Manager and iCloud Keychain), LastPass and 1Password are two of the most widely used password managers/generators [44]. LastPass permits the user to select one of three password-generation algorithms, namely "Easy-to-say", "Easy-to-read", or "All-characters". For each type, users can further specify the generator by checking or unchecking "Uppercase", "Lowercase", "Numbers", or "Symbols", though the "Easy-to-say" generator does not permit inclusion of Symbols or Numbers. 1Password allows users to

[2]We used PyAutoGui (https://pyautogui.readthedocs.io/en/latest/) to automate interactions with the password managers like 1Password and LastPass. That is, we automated generating random passwords, copying them into the clipboard, and storing them in a local file interactively.

(a) Honeyword generator $\mathcal{H}_n$ (b) Attacker $\mathcal{B}$

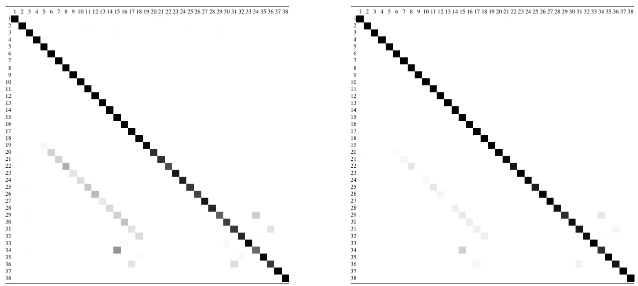

Figure 5: Confusion matrices: Probability with which a password of one class (row) is classified as another class (column) by $\mathcal{H}_n$ (Fig. 5a) or $\mathcal{B}$ with $|X| > 1$ (Fig. 5b). Box shading is scaled linearly between 0.0 (white) and 1.0 (black).

select a type of password from among "Random Password", "Memorable Password", and "Pin". The Random Password generator includes Lowercase and Uppercase letters always, but users can check or uncheck Numbers and Symbols. The Memorable Password algorithm generates memorable passwords, each of which is a sequence of word fragments connected by separators. In this option, users can select separators among "Hyphens", "Spaces", "Periods", "Commas", "Underscores", "Numbers", and "Numbers and Symbols". In addition, users could check or uncheck "Full Words" and "Capitalize" to specify the "Memorable Password" generator. In this work, we used all the configurations from LastPass and 1Password's Random Password, and selected configurations for 1Password's Memorable Password. We consider passwords generated from each specification as one class, yielding 38 classes in total. These classes are shown in Table 3. We set the fixed $\mathcal{U}_{y_{\mathsf{fx}}}$ to be the "All characters" generator from LastPass with "U", "L", "S", and "N" checked ($y_{\mathsf{fx}} = 32$).

Using these online generators, we generated three datasets, denoted $D^{\mathsf{tr}}_{\mathsf{a}}$, $D^{\mathsf{va}}_{\mathsf{a}}$, and $D^{\mathsf{te}}_{\mathsf{a}}$, all consisting of passwords of length 14 only. We used $D^{\mathsf{tr}}_{\mathsf{a}}$ to train a classifier to classify random passwords and evaluated it on $D^{\mathsf{va}}_{\mathsf{a}}$. To assemble $D^{\mathsf{tr}}_{\mathsf{a}}$ and $D^{\mathsf{va}}_{\mathsf{a}}$, we generated 8,000 and 2,000 passwords from each class, yielding 304,000 and 76,000 passwords in total, respectively. We applied $D^{\mathsf{te}}_{\mathsf{a}}$ in the evaluation of FPP and FNP. In $D^{\mathsf{te}}_{\mathsf{a}}$, there were 38 classes of passwords, each containing 10,000 sets (corresponding to 10,000 users) with 100 passwords of that class. When evaluating FPP and FNP, we implemented $\mathcal{U}$ by sampling p and X without replacement from a set (user) chosen uniformly at random from $D^{\mathsf{te}}_{\mathsf{a}}$.

5.3.2 Experimental Results

We evaluated the honeyword-generation methods described in Sec. 5.2, though we plot only $\mathsf{FNP}(\mathcal{B})$ since $\mathsf{FPP}(\mathcal{A})$ was essentially perfect. We plot $\mathsf{FNP}(\mathcal{B})$ against α in Fig. 6. As seen there, both the FXED and RAND methods had a high

Class index	Manager	Type	Alphabet U	L	S	N
1	LastPass	Easy to say		✔		
2	LastPass	Easy to say	✔			
3	LastPass	Easy to say	✔	✔		
4	LastPass	Easy to read			✔	
5	LastPass	Easy to read				✔
6	LastPass	Easy to read			✔	✔
7	LastPass	Easy to read		✔		
8	LastPass	Easy to read		✔	✔	
9	LastPass	Easy to read		✔		✔
10	LastPass	Easy to read		✔	✔	✔
11	LastPass	Easy to read	✔			
12	LastPass	Easy to read	✔		✔	
13	LastPass	Easy to read	✔			✔
14	LastPass	Easy to read	✔		✔	✔
15	LastPass	Easy to read	✔	✔		
16	LastPass	Easy to read	✔	✔	✔	
17	LastPass	Easy to read	✔	✔		✔
18	LastPass	Easy to read	✔	✔	✔	✔
19	LastPass	All characters				✔
20	LastPass	All characters			✔	✔
21	LastPass	All characters		✔		
22	LastPass	All characters		✔	✔	
23	LastPass	All characters		✔		✔
24	LastPass	All characters		✔	✔	✔
25	LastPass	All characters	✔			
26	LastPass	All characters	✔		✔	
27	LastPass	All characters	✔			✔
28	LastPass	All characters	✔		✔	✔
29	LastPass	All characters	✔	✔		
30	LastPass	All characters	✔	✔	✔	
31	LastPass	All characters	✔	✔		✔
32	LastPass	All characters	✔	✔	✔	✔
33	1Password	Random Password	✔	✔	✔	
34	1Password	Random Password	✔	✔		
35	1Password	Random Password	✔	✔	✔	✔
36	1Password	Random Password	✔	✔		✔
37	1Password	Memorable Password		✔		✔
38	1Password	Memorable Password		✔	✔	✔

Table 3: Classes of algorithmically generated passwords used in our experiments

$\mathsf{FNP}(\mathcal{B})$. Even when $\alpha = 1$, they had $\mathsf{FNP}(\mathcal{B}) > 0.94$ for $n = 19$ and $\mathsf{FNP}(\mathcal{B}) > 0.93$ for $n = 99$.

In contrast, the CLSD method achieves nearly perfect $\mathsf{FNP}(\mathcal{B})$. This method selects the most plausible algorithmic password generator based on the account password to generate honeywords. The confusion matrix experienced by $\mathcal{H}_n$ (i.e., using p) are shown in Fig. 5a. When $|X| = 1$, the confusion experienced by $\mathcal{B}$ is virtually identical, of course, but the confusion experienced by $\mathcal{B}$ when $|X| > 1$ is notably less, as shown in Fig. 5b. As this figure shows, when $|X| > 1$, $\mathcal{B}$ has greater ability to classify the user's password generator based on X than $\mathcal{H}_n$ does based on p, at least for certain classes. Since our dataset is dominated by accounts for which the number of passwords known by $\mathcal{B}$ numbers $|X| = 1$ (Fig. 2), the confusion shown in Fig. 5a (where $|X| > 1$) cannot effectively be exploited by $\mathcal{B}$.

However, if the fraction of accounts for which $\mathcal{B}$ holds

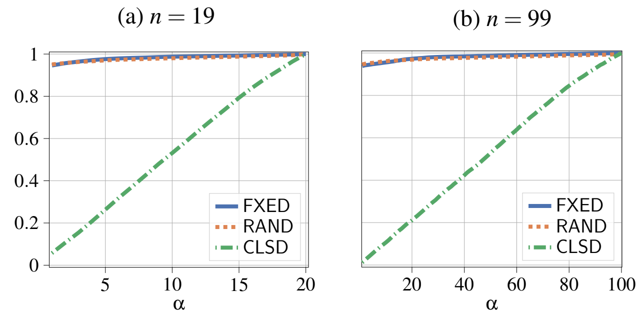

Figure 6: $\mathsf{FNP}(\mathcal{B})$ for honeywords by algorithmic password generators

$|X| > 1$ passwords were larger, the better classification accuracy this would enable (Fig. 5b) would permit an average increase in $\mathsf{FNP}(\mathcal{B})$. To illustrate this, in Fig. 7 and Fig. 8 we show the effect on $\mathsf{FNP}(\mathcal{B})$ of increasing $|X|$ from its original distribution to $|X| = 99$ always, for $n = 19$ or $n = 99$, respectively. Each subfigure shows $\mathsf{FNP}(\mathcal{B})$ for certain classes of the actual password p; e.g., Fig. 7a shows this effect when $p \leftarrow \mathcal{U}_{22}()$. As can be seen in these figures, increasing $|X|$ to $|X| = 99$ enables $\mathcal{B}$ to improve $\mathsf{FNP}(\mathcal{B})$, for these classes noticeably.

In conclusion, in the case of algorithmically generated passwords, it is critical for $\mathcal{H}_n$ to identify the algorithmic password generator used by each user in order to achieve low $\mathsf{FNP}(\mathcal{B})$. Even then, as the number of passwords $|X|$ grows, this measure will decay.

6 Discussion

6.1 False-Negative Attacks with Less Auxiliary Information

In Sec. 4, we assumed that the false-negative attacker has knowledge of the passwords used by the same user at other sites. We additionally explored a setting where the false-negative attacker had minimal auxiliary information about the users, specifically only one password used by the same user at another site ($|X| = 1$). To do so, we implemented the algorithm $\mathcal{U}$ by choosing two passwords without replacement from a single multiset S_{addr} chosen uniformly at random from $D_{\mathsf{u}}^{\mathsf{te}}$, returning one as p and the other as the only element of X. The experimental results in this setting are shown in Fig. 9. When the false-negative attacker $\mathcal{B}$ had little information about users, his success rate of guessing the account passwords from sweetwords slightly dropped, yielding a smaller $\mathsf{FNP}(\mathcal{B})$. However, all the honeyword-generation methods remained insufficiently resilient to these attacks. The best $\mathsf{FNP}(\mathcal{B})$ that the honeyword-generation techniques accomplished for all accounts was 0.48 (P2P, Fig. 9r). No existing algorithm achieved low rates of both false positives and false negatives.

6.2 Countermeasures to False-Positive Attacks

False-positive attacks can be very costly to sites, if they induce an investigation into the possibility of a breach and/or a password reset for every site account. Moreover, repeated false positives might eventually result in the defense being ignored or disabled outright. Despite the consequences of false positives, only a few previous works on honeywords have briefly discussed how to prevent them [22, 48]. Wang, et al. [48] suggested that applying a blocklist of popular passwords to honeyword selection can reduce false positives, since the honeyword-generation methods considered in their work generate honeywords by sampling from a public password distribution (e.g., leveraging a password model like List). As such, a blocklist would avoid using popular passwords as honeywords, which can mitigate the guessability of honeywords by their proposed methods. However, a blocklist of popular passwords is much less effective when considering password-dependent honeyword-generation algorithms (e.g., CBT, CHM, Tweak, and P2P), since these methods assign more likelihood to those candidates similar to the account password. A way to mitigate false positives of these methods is to avoid using passwords similar to the account password as honeywords, which makes them suffer a high false-negative rate.

Another countermeasure to reduce false positives, as mentioned by Juels and Rivest [22], is to select n honeywords uniformly at random from a large pool of candidate honeywords that are similar to the account password. In order to achieve a small false-positive rate, the size of the pool should be much larger than n. However, it is challenging to generate such a large pool of candidates that are sufficiently similar to the account password to ensure a small false-negative rate via this process. As such, an interesting direction is to explore how to generate such a large candidate pool to achieve a target false-negative rate.

Ultimately, a site might find it most palatable to address the risk of false positives by adopting a lenient policy toward honeyword-induced breach alarms. Previous works (e.g., [22, Sec. 2.4]) outlined a range of possible reactions to honeyword-induced alarms, ranging from severe (e.g., shutting down the system and forcing all users to reset their passwords) to lenient (e.g., allow the login to proceed as usual). Whatever the policy, however, it will apply to both false and true positives alike, and so a policy can be relaxed only so far as is acceptable when a breach actually occurs.

6.3 Balancing Attention to False Positives and False Negatives

Since honeywords' proposal, a challenge has been to design good honeyword-generation methods that achieve both low false-positives and low false-negatives, i.e., $\mathsf{FPP} \approx 0$ and $\mathsf{FNP} = \frac{\alpha}{n+1}$. However, our experimental results in Sec. 4.3.2

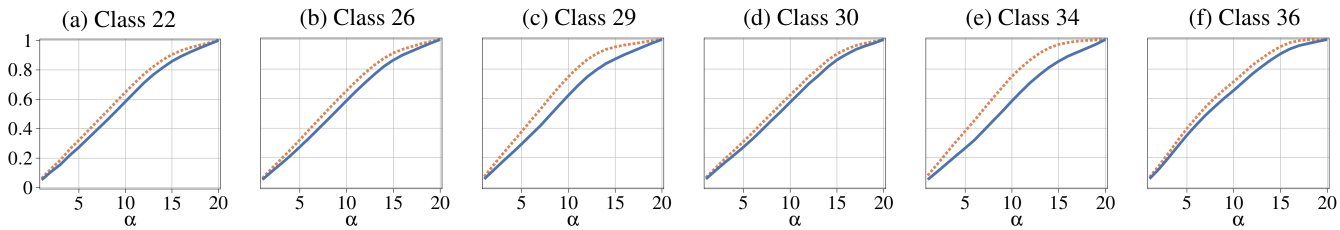

Figure 7: $\mathsf{FNP}(\mathcal{B})$ for $\mathcal{H}_n$ of CLSD ($n = 19$), for $|X|$ distributed as in Fig. 2 (–) and for $|X| = 99$ (--)

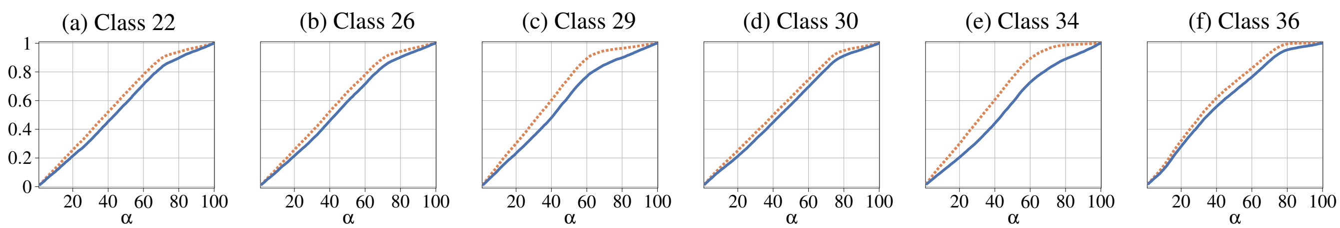

Figure 8: $\mathsf{FNP}(\mathcal{B})$ for $\mathcal{H}_n$ of CLSD ($n = 99$), for $|X|$ distributed as in Fig. 2 (–) and for $|X| = 99$ (--)

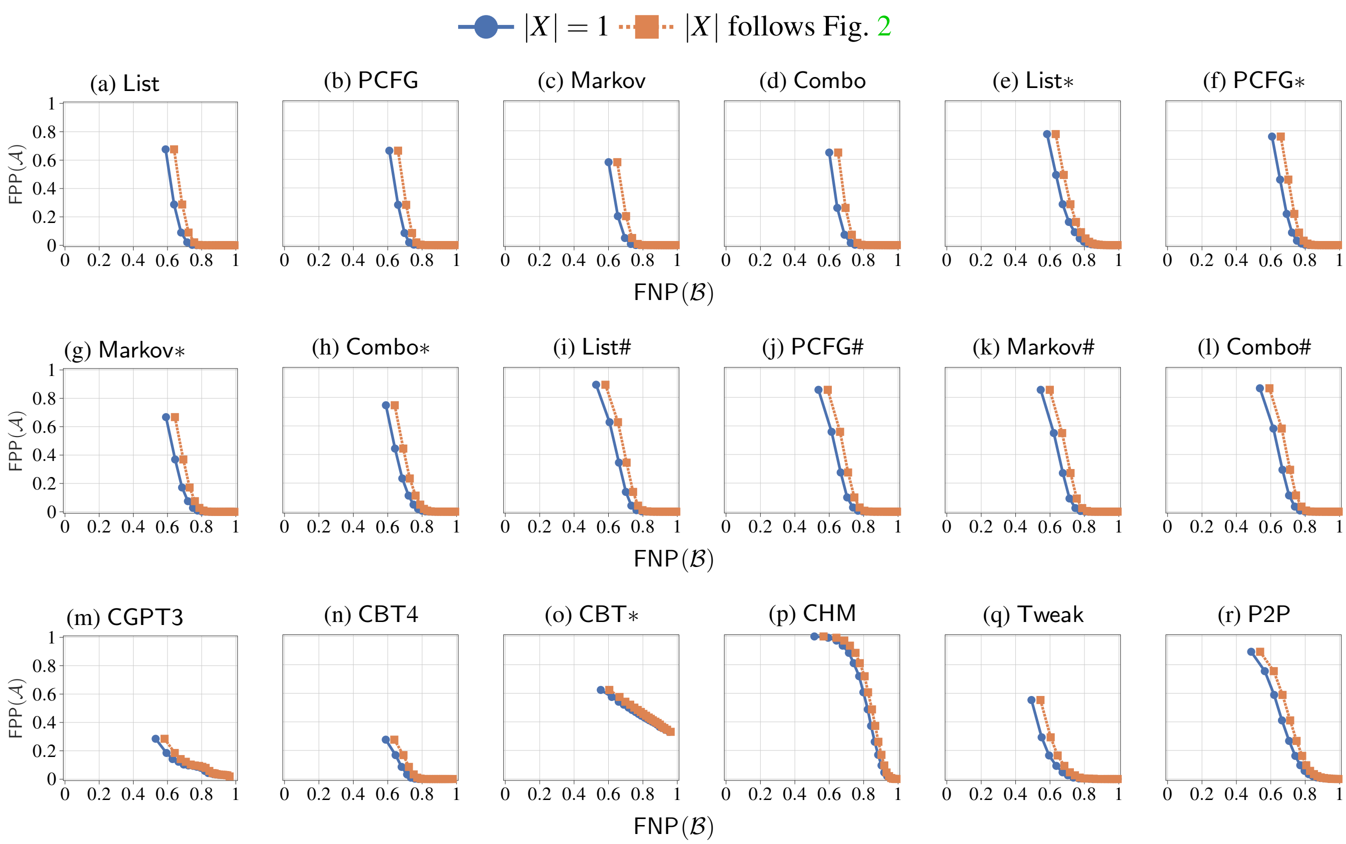

Figure 9: $\mathsf{FPP}(\mathcal{A})$ vs. $\mathsf{FNP}(\mathcal{B})$ as $|X|$ is varied, for the case of user-chosen passwords ($n = 19$, $\beta = 1000$) and all accounts.

show that no existing method achieves this goal in a threat model in which passwords from the same user at other sites are exposed to the attacker. While this leaves us skeptical that a perfect honeyword-generation method exists for this threat model (at least not when passwords are user-chosen, versus algorithmically generated), we do not mean to suggest that research in this direction should end. However, we do advocate that new honeyword-generation methods should be investigated with balanced attention to false positives and false negatives in this threat model, rather than more narrowly focusing on false negatives, as has been typical in most prior research.

In the absence of improved honeyword-generation algorithms that more effectively balance false positives and false negatives, we believe that a change in perspective on the use of honeywords might be warranted. One such perspective, reflected in recent work [53], is to tightly constrain false positives to enable a honeyword-induced alarm to confidently be taken seriously and dealt with severely, and then to rely on a false-negative attacker's interests in harvesting *many* accounts to likely trigger a breach alarm even if the true-positive probability *per account* is modest. Such an approach will likely permit breaching attackers to harvest selected accounts without triggering a breach alarm, but greedy breaching attackers will still trigger a breach alarm with high probability.

Site	Password composition policy
google.com	$\geq$ 8 characters including a letter, a symbol, and a number
youtube.com	$\geq$ 8 characters including a letter, a symbol, and a number
facebook.com	$\geq$ 6 characters including a letter, a symbol, and a number
twitter.com	$\geq$ 8 characters including a letter, a symbol, and a number
instagram.com	$\geq$ 6 characters including a letter, a symbol, and a number
baidu.com	$\geq$ 8 and $\leq$ 14 characters of at least two types from uppercase letter, lowercase letter, symbol, and number
wikipedia.org	$\geq$ 8 characters
yandex.ru	$\geq$ 6 characters including an uppercase letter, a lowercase letter, and a number
yahoo.com	$\geq$ 9 characters
xvideos.com	No requirement
pornhub.com	$\geq$ 6 characters
amazon.com	$\geq$ 6 characters
tiktok.com	$\geq$ 8 and $\leq$ 20 characters including a letter, a symbol, and a number
live.com	$\geq$ 8 characters of at least two types from uppercase letter, lowercase letter, symbol, and number
openai.com	$\geq$ 8 characters
reddit.com	$\geq$ 8 characters
linkedin.com	$\geq$ 6 characters
netflix.com	$\geq$ 6 characters
office.com	$\geq$ 8 characters of at least two types from uppercase letter, lowercase letter, symbol, and number
twitch.tv	$\geq$ 8 characters

Table 4: Most visited websites [36] and their password composition policies retrieved in May 2023.

Tranco Top	U	L	S	N
10K	81.7%	78.4%	80.3%	71.2%
100K	79.4%	79.2%	76.3%	82.5%
1M	83.7%	84.1%	86.3%	82.0%

Table 5: Percentages of sites from Tranco Top 10K, 100K, and 1M that do not require some types of characters (U: uppercase letter, L: lowercase letter, S: special symbol, N: number). These statistics were obtained on Dec. 2021 [3].

6.4 Assumptions on Algorithmic Password Generator Configuration

A limitation of our analysis in Sec. 5 is that it was conducted assuming that the user uniformly randomly selects a configuration for her algorithmic password generator and, once adopting a configuration, does not change it.

Changes of password manager configuration: To justify the assumption that users rarely change password manager configurations, we performed a study of the password-creation policies of twenty commonly accessed websites and those from Tranco Top 1M websites [32] to show that users are rarely *required* to change configurations. Specifically, we sought to determine the frequency with which a user who sets passwords at these sites in a random order will be required to change his password-creation algorithm configuration to comply with the next site in the sequence.

To perform this evaluation, we retrieved the password requirements from twenty commonly visited websites [36], shown in Table 4, or simulated password composition policies based on the statistics from a recent large-scale study [3], shown in Table 5. For a sequence of password policies that is a permutation of password requirements from the twenty commonly visited websites or a sequence of 101 simulated password-composition policies randomly constructed from the statistics, we evaluated the number of times that the current password-generator configuration conflicted with the password-creation policy of the next website in the sequence, starting from a configuration initialized by the minimum password requirement of the first site. To ensure a conservative evaluation, when a conflict occurred, the current configuration was replaced with the minimum password requirements of the conflicting site. For each type of password policy sequence, we performed this analysis for 10^6 times. The evaluation results are shown in Table 6. We found that the numbers of conflicts ranged from an average of 2.143 in sequences of 20 websites to 15.829 in sequences of 101 simulated sites drawn from Tranco Top 10,000 sites, averaged over the 10^6 sequences. These implied a probability of conflict with the next site in the sequence between 0.1127 and 0.1529. In expectation, then, the probability of exactly κ consecutive resets with no conflicts, followed by a site that conflicts, is $< (1-0.1529)^{\kappa}(0.1529)$, and the average number of non-

Sites	Number of conflicts	Probability of conflict with the next site	Average number of non-conflicting sites before a conflict
20 commonly visited websites	2.143	0.1127	8.864
101 simulated sites from Tranco Top 10K	15.829	0.1529	6.317
101 simulated sites from Tranco Top 100K	14.450	0.1445	6.920
101 simulated sites from Tranco Top 1M	11.825	0.1182	8.456

Table 6: Evaluation results on random walking at websites.

conflicting sites before a conflict was > 6.3. Given the conservative nature of our evaluation, we believe this result justifies our assumption that a user would rarely change its password-generator configuration.

However, an interesting direction for future work would be to confirm or refute this assumption more broadly, since as shown in Sec. 5.3.2, the assumption somewhat diminishes the effectiveness of honeywords generated for accounts with algorithmically generated passwords. Alternatively, an algorithmic password generator could be designed to encourage changing these configuration settings regularly, in which case an interesting research direction would be to explore the acceptability of this practice for users.

Tendency to use default password configuration: In reality, we expect people to generally defer to the default password-manager configuration until forced to change it by a site's password policy. However, imposing a non-uniform distribution on $\{\mathcal{U}_y\}_{y=1}^{Y}$ to reflect this tendency should not qualitatively change the results of our study: First, analogous to our results in the full paper [20, App. E], the existing honeyword-generation methods would still fail to provide a low false-negative probability since the false-negative attacker would even more easily distinguish the algorithmically generated password from the honeywords if he knows how the selection of configuration is biased. Second, as shown in Sec. 5.3.2, when the honeyword system can correctly predict the configuration used by the user—which should only become easier when the distribution is biased toward the default—and use that configuration to generate the honeyword, the generated honeywords can provide sufficient security.

6.5 A Mixed Case Study

In this work, we studied two representative cases where users create user-chosen passwords (Sec. 4) or where users generate their passwords algorithmically using a password manager (Sec. 5). To assess the efficacy of honeywords when users employ mixed strategies (i.e., chose some passwords themselves and algorithmically generate others), we further constructed two test datasets by mixing $D_{\mathsf{u}}^{\mathsf{te}}$ and the algorithmically generated dataset. Then we generated honeywords based on the type of the account password, i.e., applying honeyword-generation methods described in Sec. 3.2 to generate honeywords for user-chosen passwords and password managers to generate honeywords for algorithmically generated passwords. Our study showed that increased use of password managers in password creation can ease the tensions brought on by password reuse and thus make better trade-offs between false-positive and false-negative rates of honeywords. More details on the experiments and results are shown in the full paper [20, App. F].

6.6 Password Reuse

Our findings that password reuse across sites is so detrimental to honeyword false-negative rates (Sec. 4.3.2) provides yet more evidence that moving more users toward password managers would be good policy (notwithstanding the risk of password-manager breaches, e.g., [39]). That said, a recent university survey [25] found that though a large majority (77%) of respondents reported using a password manager, another large majority (again, 77%) also reported still reusing passwords across accounts. So, while a step in the right direction, password managers are evidently not a panacea. A potentially more effective approach might be explicitly hindering attempts to reuse passwords, either through adoption of intentionally conflicting password requirements at websites (which is not commonplace, see Sec. 6.4) or through explicit interventions during the password (re)setting process to interfere with reusing the same or similar passwords (e.g., [51]).

7 Conclusion

In this paper, we have conducted the first critical analysis of honeyword-generation techniques for users who have suffered exposed passwords for their accounts at other sites. We formalized the false-positive rate and false-negative rate of honeywords in a model where the attacker has access to passwords for the same users at other sites or, in the case of false-positive attackers, even passwords for users at the defending site (as the real users would). Using these formalized definitions and a large dataset of leaked passwords, we experimentally demonstrated that existing honeyword-generation algorithms exhibit poor tradeoffs between false positives and false negatives when the account password is chosen by an average human user. Then we studied the case where the account password is algorithmically generated and used passwords from popular password managers to show that the existing honeyword-generation methods offer modest protection against false-negative attackers. We further explored the use of algorithmic password generators in honeyword generation and determined that seemingly the only effective strategy is to generate honeywords using the same password generator that the user does, if it can determine what that password generator is. In total, we believe our results paint a cautionary picture for the state of honeyword-generation algorithms to date, though they also set forth new research challenges for the field.

Acknowledgements

The work described in this paper was supported in part by NSF grants 2113345 and 2131859. We are also grateful to Bhuwan Dhingra for helpful discussions.

References

[1] Akshima, D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya. Generation of secure and reliable honeywords, preventing false detection. *IEEE Transactions on Dependable and Secure Computing*, 16(5):757–769, 2019.

[2] M. H. Almeshekah, C. N Gutierrez, M. J. Atallah, and E. H. Spafford. ErsatzPasswords: Ending password cracking and detecting password leakage. In *31st Annual Computer Security Applications Conference*, pages 311–320, December 2015.

[3] S. Alroomi and F. Li. Measuring website password creation policies at scale. In *30th ACM Conference on Computer and Communications Security*, 2023.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 2020.

[5] J. Casal. 1.4 billion clear text credentials discovered in a single database. https://medium.com/4iqdelvedeep/1-4-billion-clear-text-credentials-discovered-in-a-single-database-3131d0a1ae14, 2017.

[6] N. Chakraborty, J. Li, V. CM Leung, S. Mondal, Y. Pan, C. Luo, and M. Mukherjee. Honeyword-based authentication techniques for protecting passwords: A survey. *ACM Computing Surveys*, 55:1–37, 2022.

[7] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *37th International Conference on Machine Learning*, 2020.

[8] N. Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3), 1956.

[9] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. The tangled web of password reuse. In *ISOC Network and Distributed System Security Symposium*, 2014.

[10] M. Dell'Amico and M. Filippone. Monte carlo strength evaluation: Fast and reliable password checking. In *22nd ACM Conference on Computer and Communications Security*, 2015.

[11] Digital Shadows Photon Research Team. From exposure to takeover: The 15 billion stolen credentials allowing account takeovers. https://resources.digitalshadows.com/whitepapers-and-reports/from-exposure-to-takeover, 2020.

[12] A. Dionysiou and E. Athanasopoulos. Lethe: Practical data breach detection with zero persistent secret state. In *7th IEEE European Symposium on Security and Privacy*, June 2022.

[13] A. Dionysiou, V. Vassiliades, and E. Athanasopoulos. Honeygen: generating honeywords using representation learning. In *16th ACM Symposium on Information, Computer and Communications Security*, 2021.

[14] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. Omen: Faster password guessing using an ordered markov enumerator. In *7th International Symposium on Engineering Secure Software and Systems*, 2015.

[15] I. Erguler. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Parallel and Distributed Systems*, 13(2), 2016.

[16] D. Florêncio, C. Herley, and P. C. van Oorschot. An administrator's guide to internet password research. In *28th Large Installation System Administration Conference*, pages 35–52, November 2014.

[17] M. Golla and M. Dürmuth. On the accuracy of password strength meters. In *25th ACM Conference on Computer and Communications Security*, October 2018.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11), 2020.

[19] X. He, H. Cheng, J. Xie, P. Wang, and K. Liang. Passtrans: An improved password reuse model based on transformer. In *47th IEEE International Conference on Acoustics, Speech and Signal Processing*, 2022.

[20] Z. Huang, L. Bauer, and M. K. Reiter. The impact of exposed passwords on honeyword efficacy. *arXiv*, 2309.10323, March 2024.

[21] IBM Security. Cost of a data breach report 2022. https://www.ibm.com/security/data-breach, 2022.

[22] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *20th ACM Conference on Computer and Communications Security*, November 2013.

[23] 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 *33rd IEEE Symposium on Security and Privacy*, 2012.

[24] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *35th IEEE Symposium on Security and Privacy*, 2014.

[25] P. Mayer, C. W. Munyendo, M. L. Mazurek, and A. J. Aviv. Why users (don't) use password managers at a large educational institution. In *31st USENIX Security Symposium*, August 2022.

[26] 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*, 2016.

[27] Microsoft. Store passwords using reversible encryption. https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/store-passwords-using-reversible-encryption, December 2022.

[28] A. Mukherjee, K. Murali, S. K. Jha, N. Ganguly, R. Chatterjee, and M. Mondal. MASCARA: Systematically generating memorable and secure passphrases. In *18th ACM Symposium on Information, Computer and Communications Security*, 2023.

[29] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *40th IEEE Symposium on Security and Privacy*, 2019.

[30] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti. Improving password guessing via representation learning. In *42nd IEEE Symposium on Security and Privacy*, 2021.

[31] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *24th ACM Conference on Computer and Communications Security*, October 2017.

[32] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *25th ISOC Network and Distributed System Security Symposium*, February 2018.

[33] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE Signal Processing Magazine*, 3(1), 1986.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088), 1986.

[35] Shape Security. 2018 credential spill report. https://info.shapesecurity.com/rs/935-ZAM-778/images/Shape_Credential_Spill_Report_2018.pdf, 2018.

[36] Similarweb. Top websites ranking. https://www.similarweb.com/top-websites/, 2023.

[37] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[38] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. Data breaches, phishing, or malware? Understanding the risks of stolen credentials. In *24th ACM Conference on Computer and Communications Security*, 2017.

[39] K. Toubba. Security incident update and recommended actions. https://blog.lastpass.com/2023/03/security-incident-update-recommended-actions/, 1 March 2023.

[40] 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*, 2015.

[41] P. C. van Oorschot. *Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin*. Springer, 2nd edition, 2021.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *30th Advances in Neural Information Processing Systems*, 2017.

[43] Verizon. 2022 data breach investigations report. https://www.verizon.com/business/resources/reports/dbir/, 2022.

[44] A. Vigderman. Password manager annual report 2022. https://www.security.org/digital-safety/password-manager-annual-report/, 27 January 2023.

[45] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian. Zipf's law in passwords. *IEEE Transactions on Information Forensics and Security*, 12(11):2776–2791, November 2017.

[46] D. Wang, H. Cheng, P. Wang, J. Yan, and X. Huang. A security analysis of honeywords. In *25th ISOC Network and Distributed System Security Symposium*, February 2018.

[47] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang. Targeted online password guessing: An underestimated threat. In *23rd ACM Conference on Computer and Communications Security*, 2016.

[48] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang. How to attack and generate honeywords. In *43rd IEEE Symposium on Security and Privacy*, May 2022.

[49] D. Wang, Y. Zou, Y. Xiao, S. Ma, and X. Chen. PASS2EDIT: A multi-step generative model for guessing edited passwords. In *32nd USENIX Security Symposium*, 2023.

[50] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin. Deep metric learning with angular loss. In *31st IEEE International Conference on Computer Vision*, 2017.

[51] K. C. Wang and M. K. Reiter. How to end password reuse on the web. In *26th ISOC Network and Distributed System Security Symposium*, February 2019.

[52] K. C. Wang and M. K. Reiter. Using Amnesia to detect credential database breaches. In *30th USENIX Security Symposium*, August 2021.

[53] K. C. Wang and M. K Reiter. Bernoulli honeywords. In *31st ISOC Network and Distributed System Security Symposium*, 2024.

[54] R. Wash, E. Rader, R. Berman, and Z. Wellmer. Understanding password choices: How frequently entered passwords are re-used across websites. In *12th Symposium on Usable Privacy and Security*, 2016.

[55] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *30th IEEE Symposium on Security and Privacy*, 2009.

[56] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han. Chunk-level password guessing: Towards modeling refined password composition representations. In *28th ACM Conference on Computer and Communications Security*, November 2021.

[57] M. Xu, J. Yu, X. Zhang, C. Wang, S. Zhang, H. Wu, and W. Han. Improving real-world password guessing attacks via bi-directional transformers. In *32nd USENIX Security Symposium*, 2023.

[58] F. Yu and M. V. Martin. Honey, I chunked the passwords: Generating semantic honeywords resistant to targeted attacks using pre-trained language models. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2023.