# PassMine: Learning to Identify Real Passwords Among Honeywords

**Anonymous ACL submission**

## Abstract

"Honeywords" are decoy passwords designed to detect password breaches by triggering alarms when used for login. The security of honeywords relies on the assumption that the real passwords and honeywords are indistinguishable to attackers.

In this study, we propose *PassMine*, a novel deep learning (DL)-based framework designed to identify real passwords from a set of sweetwords, which includes one real password mixed with multiple honeywords. This approach effectively challenges a range of existing and potentially future honeyword generation techniques (HGTs). Using character-level convolutional neural network trained on public password leaks, PassMine is able to distinguish real passwords from honeywords produced using state-of-the-art HGTs. Our results on nine public datasets demonstrate that PassMine achieves a ≈53% success rate on the first attempt (vs. 5% random guessing) and ≈99% within ten attempts (vs. 50% random guessing). Moreover, PassMine substantially outperforms the prior state-of-the-art attack of Huang et al. (2024) while requiring no per-user cross-site auxiliary passwords. These findings highlight significant vulnerabilities in current HGTs to DL-based attacks.

## 1 Introduction

Passwords are a commonly used mechanism to authenticate and confirm a user's identity when logging into a remote service. Typically, servers store passwords in a hashed form to protect them from unauthorized access. However, passwords remain vulnerable to threats such as phishing, credential stuffing driven by password reuse, online guessing, and server-side breaches. In particular, when an attacker obtains a leaked database of password hashes (e.g., salted hashes), they can mount offline dictionary attacks (ODAs) to recover plaintext passwords. Indeed, over the years, many popular web services have been compromised, resulting in the leakage of millions of users' data and passwords. For example, well-known services have been compromised (Hackett, 2021; O'Sullivan, 2024; security news By Steve Ragan and Ragan, 2016; Cybercrime Magazine, 2024), leading to the disclosure of sensitive user information, including passwords. This situation has underscored the importance of implementing more secure authentication mechanisms and better password storage practices to minimize the risks of such breaches.

Traditionally, system administrators employ fake, decoy or falsified accounts as an early warning system to detect data breaches, but this practice is facing new challenges. Despite serving as a preemptive measure, the efficacy of such accounts is diminishing due to the escalating sophistication of malicious actors who can promptly identify them, especially through scrutiny of usernames (Juels and Rivest, 2013). In response to this evolving threat landscape, Juels and Rivest proposed employing "honeywords" to bolster password security. Unlike conventional fake accounts, honeywords provide a cost-effective strategy to strengthen authentication systems, making them an attractive choice for organizations seeking robust security enhancements.

The fundamental principle of honeywords is to store for each account multiple decoy passwords alongside the real password, forming a list of sweetwords. Depending on the honeyword-generation method, decoys may be constructed to resemble user-chosen passwords. A separate "honeychecker" stores the index of the real password and raises an alarm if an online login succeeds using a decoy; thus, after a database compromise, an attacker must distinguish the real password from the decoys.

In this study, we address the following research question: *Can we distinguish between passwords and honeywords generated using state-of-the-art and emergent techniques, including the representation learning based approaches proposed by Diony-*

*siou et al. (2021)?* We answer this question affirmatively. We introduce the PassMine attack, a DL-based method that is fundamental in its ability to efficiently extract a user's password from the list of sweetwords linked to the user's account. This approach is designed to be robust against a variety of state-of-the-art HGTs, ensuring its relevance and effectiveness in evolving security landscapes. We benchmark against random guessing and the prior state-of-the-art attack by Huang et al. (2024).

Our work bridges security and natural language processing (NLP). We treat passwords as short, language-like character sequences, applying sequence modeling to distinguish between real and synthetic (algorithmically generated) strings. Much like synthetic text detection in NLP (Gehrmann et al., 2019; Ippolito et al., 2020; Huang et al., 2025), the task of distinguishing passwords from honeywords can be viewed as a special case of sequence classification, where the goal is to discriminate genuine human-generated sequences from synthetic ones. By framing password identification as a sequence modeling problem, our work highlights new vulnerabilities at the intersection of NLP and cybersecurity, and underscores the broader impacts of advances in neural language modeling for security applications.

Our main contributions and findings are summarized as follows:

1. **Our Methodology:** We formulate password-honeyword discrimination as binary sequence classification on short character sequences (8-20 chars). We propose a character-level CNN that separates real passwords from synthetically generated decoys across multiple HGTs, requiring only attacker knowledge of the HGT and access to public passwords leak. Unlike Huang et al. (2024), no cross-site auxiliary data or PII is needed.
2. **Comprehensive Evaluation:** We evaluate PassMine on nine leaked-password datasets (450k+ passwords) and three HGTs, in both in-distribution and cross-dataset settings. PassMine attains a ≈53% success rate on the $1^{st}$ attempt and a ≈99% success rate within 10 attempts, outperforming Huang et al. (2024). We also demonstrate robust cross-dataset generalization despite significant distributional shift from varying password policies.
3. **Empirical Insights:** Our results indicate that representation-learning-based generators produce statistically detectable patterns, heuristic perturbation methods are particularly vulnerable, and numeric-only passwords present unique challenges where models approach random performance.

**Reproducibility.** We will release code and data upon publication.

## 2 Threat Models and Assumptions

In this study, we consider a scenario where a user ($\mathcal{U}$) interacts with a web service $\mathcal{S}$ that provides a password-based authentication enhanced with honeywords to defend against password guessing. An attacker ($\mathcal{A}$) aims to impersonate the user by compromising $\mathcal{S}$ and obtaining the hashed sweetwords file, which contains both the real password and the honeywords for each account.

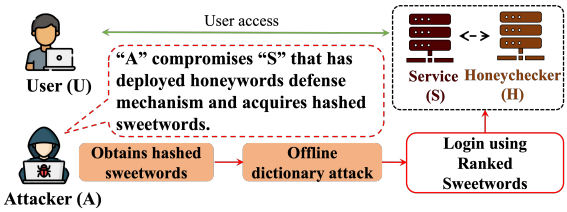



Figure 1: $\mathcal{A}$ compromises web service using honeywords defense mechanism, acquires the hashed sweetwords, perform offline dictionary attack, and uses PassMine to rank sweetwords and attempt logins.

The attacker employs offline dictionary attacks or password guessing techniques (Hitaj et al., 2019; Narayanan and Shmatikov, 2005; Liu et al., 2019; Weir et al., 2009a; Xu et al., 2023), or tools like Hashcat (Steube, 2023) or data-driven models such as Markov Chains (Ma et al., 2014a; Narayanan and Shmatikov, 2005), to efficiently recover the plaintexts of all sweetwords. We assume the attacker does not have access to the user's personally identifiable information (PII; e.g., email address). The honeychecker ($\mathcal{H}$) is a separate, hardened component responsible for determining whether a submitted sweetword is the user's real password or a honeyword.

In classic honeyword deployments, a successful login using a honeyword triggers an alarm. In practice, deployments may tolerate a small number of honeyword events (e.g., due to typos when honeywords are minor edits of the true password), and online services rate-limit authentication attempts. Accordingly, we model two attempt budgets: a per-account threshold $\mathcal{TH}_1$ and a system-wide threshold $\mathcal{TH}_2$ (e.g., escalation after repeated honeyword

events). We consider an attack successful if $\mathcal{A}$ identifies the real password within the allowed number of online attempts before triggering $\mathcal{TH}_1$ or $\mathcal{TH}_2$, selecting concrete threshold values is deployment-specific and outside the scope of this study.

We further assume the attacker knows the HGT used by the system, consistent with Kerckhoffs's principle: security should rely on the secrecy of the honeywords, not the generation algorithm. This is realistic since HGT implementations are commonly open-source (e.g., Dionysiou et al. (2021)), and an attacker can fingerprint an unknown HGT by analyzing recovered plaintexts from prior breaches (Appendix F). For training data, billions of plaintext passwords are freely available from public breach dumps (e.g., RockYou, LinkedIn), requiring no per-user information. Unlike Huang et al. (2024), who require cross-site passwords linked to the same individual user, PassMine needs only this widely available aggregate data, modeling a practical attacker without relying on privileged per-user information.

## 3 Background & Related Work

**Honeywords Security Analysis.** Honeywords, introduced by Juels and Rivest (2013), are decoy passwords inserted in the databases to detect password leakage. Several studies have analyzed their design, security, and the effectiveness of various honeyword generation strategies, revealing vulnerabilities in approaches such as simple tweaking or the "honeyindex" method (Wang et al., 2022a; Akshima et al., 2019; Erguler, 2016; Guo et al., 2021; Huang et al., 2024). We provide a comprehensive review of attacks and HGTs in Appendix A.

However, prior studies have not systematically evaluated data-driven HGTs such as those developed by Dionysiou et al. (2021). While Huang et al. (2024) evaluate some of the same HGT families (Chaffing-by-Tweaking and Chaffing-with-a-Hybrid-Model), two important distinction exists. First, Huang et al. (2024)'s tweaking generator uses a simplified replacement scheme rather than Dionysiou et al. (2021)'s data-driven, making the two evaluations not directly comparable. Second, Huang et al. (2024) frame attack as a similarity-ranking via contrastive DNN, requiring per-user cross-site auxiliary passwords. This is a more privileged attacker assumption than PassMine, which requires only public password leaks and HGT knowledge. Additionally, the Chaffing-with-a-Password-Model HGT proposed by Dionysiou et al. (2021) has not been thoroughly explored in prior work.

**Honeywords Generation Techniques (HGTs).** In this study, we consider HGTs proposed by Dionysiou et al. (2021):

- **Chaffing-by-Tweaking (CbT)** is a heuristic approach that perturbs the real passwords by substituting characters with replacement probabilities derived from data analysis of leaked passwords.
- **Chaffing-with-a-Password-Model (CbP)** employs representation learning to generate honeywords. Specifically, it uses a `FastText` embedding model trained on a large corpus to produce honeywords that are semantically and structurally similar to the real passwords. The model generates top-$k$ nearest neighbors in the embedding space as candidate honeywords.
- **Chaffing-with-a-Hybrid-Model (CbH)** combines both approaches: it first uses CbP approach to generate candidate honeywords and then further perturbs them using CbT technique. This hybrid strategy is designed to enhance the diversity and non-reversibility of generated honeywords.

The goal is to generate honeywords that are structurally and semantically similar to the real password. At the same time, they should be difficult for adversaries to distinguish. Dionysiou et al. (2021) show that these methods can achieve low success rates for CbP and CbH HGTs. Their effectiveness approaches the theoretical optimum of random guessing when tested against attack developed by Wang et al. (2018).

**Synthetic Data Detection using DL.** Distinguishing real from algorithmically generated sequences is well-studied in NLP (Gehrmann et al., 2019; Ippolito et al., 2020; Huang et al., 2025). We extend this line of research to the password security.

In the security domain, similar questions arise in password analysis. For instance, Hitaj et al. (2019) investigated whether GAN-generated passwords can be distinguished from user-created passwords. In the context of honeywords, Dionysiou et al. (2021) conducted user studies to assess whether humans can identify honeywords among real passwords. However, there has been limited systematic study of discriminative DL models for distinguishing user-generated passwords from algorithmically generated honeywords. Our work extends this line of research by applying sequence classification models to this challenge, assessing the efficacy of DL models in the password security setting.

## 4 PassMine Design & Methodology

In this section, we introduce PassMine, a password identification attack to exploit weakness in HGTs (Figure 2).

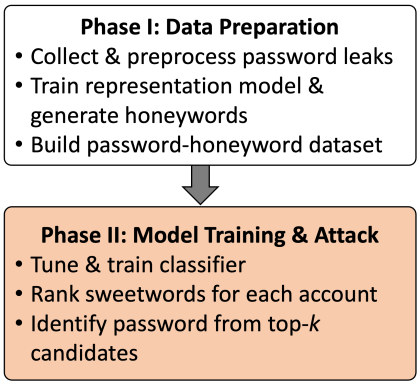



Figure 2: The design of PassMine to identify passwords from a list of sweetwords associated with a user's account. In Phase I, a dataset of honeywords corresponding to the passwords is created. In Phase II, both honeywords and passwords are tokenized to train and evaluate a CNN model.

In Phase I, the attacker ($\mathcal{A}$) first assembles a large corpus of leaked passwords (e.g., breach dumps made publicly available by hackers or insiders). This corpus supports both honeyword generation (using `FastText`) and subsequent discriminator training (a character-level CNN). During preprocessing, we standardize encoding and formatting (e.g., canonicalizing delimiters and removing parsing-breaking artifacts) while preserving the original password strings.

We adopt a character-level CNN for PassMine for several reasons. First, passwords are extremely short (8-20 characters vs. 20-500 tokens in typical text classification), where character n-grams effectively capture both semantic patterns (e.g., "Password") and substitutions (e.g., "P@ssw0rd"). Second, since honeywords are designed to mimic genuine passwords, discriminative models are needed to learn subtle distinguishing features. Third, CNNs have proven effective in security classification (Wagh et al., 2013; Kambar et al., 2022; Dani and Wang, 2021; Das et al., 2019; Iqbal et al., 2021; Roseline et al., 2022). We adopt a character-level CNN primarily for its computational efficiency: CNNs process sequences in parallel rather than sequentially, making them significantly faster to train than LSTMs at scale (LeCun et al., 2015; Weytjens and De Weerdt, 2020). Given that our NAS procedure runs up to 200 trials per dataset–HGT pair, this efficiency is critical. Whether recurrent or attention-based architectures yield higher discriminative accuracy on this specific task remains an open question for future work.

Following preprocessing, $\mathcal{A}$ trains a `FastText` model in an unsupervised manner on the collected password corpus (Bojanowski et al., 2016) and uses it to generate honeywords for a given input password, following Dionysiou et al. (2021). Additionally, $\mathcal{A}$ instantiates the heuristic generators described in Dionysiou et al. (2021) to produce complementary honeyword sets.

In Phase II (featurization, model training, and attack execution), $\mathcal{A}$ trains a character-level CNN discriminator to distinguish passwords from honeywords. Given an input string $s = c_1 c_2 \ldots c_n$, we map characters to learned embeddings, $c_1 c_2 \ldots c_n \mapsto (v_1, \ldots, v_n), \quad v_i = E(c_i)$, with padding to a fixed maximum length.

The encoder comprises a stack of convolutional blocks followed by fully connected blocks; the number of blocks and other architectural/training hyperparameters are selected via tuning (Section 5). Each convolutional block applies a 1D convolution followed by batch normalization, a pointwise nonlinearity ($\phi$), pooling, and dropout: $x_\ell = \mathrm{Dropout}\Big(\mathrm{Pool}\big(\phi(\mathrm{BN}(\mathrm{Conv}(x_{\ell-1})))\big)\Big)$, and each fully connected block is $h_j = \mathrm{Dropout}\big(\phi(\mathrm{BN}(W_j h_{j-1} + b_j))\big)$. The final layer produces a binary prediction (2-way softmax) and we train using categorical cross-entropy loss. At inference time, given a sweetword set for an account, $\mathcal{A}$ scores each candidate with the discriminator and selects the highest-scoring string as its password guess. The complete architecture specification is shown in Fig. 8 (Appendix B). Further, we provide details on hyperparameter tuning in Section 5.

We leverage this trained CNN model to compute $\epsilon$-flatness, conducting a sophisticated analysis of sweetwords' probability scores for password identification. For computing $\epsilon$-flatness, consider a collection of sweetwords $\mathcal{X}_i$ associated with the user $\mathcal{U}_i$. Each sweetword $\mathcal{X}_{ik}$ ($1 \leq k \leq 20$) is provided as input to the classifier, and obtained the likelihood (probability score) of each of a given $\mathcal{X}_{ik}$ is real password. The details for ranking sweetwords in discussed in Section 5.3.

## 5 Evaluation Framework

In this section, we describe the datasets, experiment settings, and evaluation metrics.

### 5.1 Description of the Datasets

We assessed the efficacy of the PassMine attack using publicly available password datasets provided by Dionysiou et al. (2021) [1]. Similar datasets have been instrumental in prior studies (Hitaj et al., 2017; Dionysiou et al., 2021; de Carné de Carnavalet and Mannan, 2014; Ma et al., 2014b). These datasets consist of passwords from various compromised web services (listed in Appendix C), publicly released by hackers or insiders. Each dataset contains 50,000 passwords, except RockYou-large which contains 1.04 million passwords.

Dionysiou et al. (2021) preprocessed these datasets by excluding passwords shorter than eight characters. Additionally, the RockYou dataset was provided in two versions, referred to as RockYou-large and RockYou-small, differing in size and sample selection criteria.

The honeywords utilized for training the classifiers were generated using the implementation provided by Dionysiou et al. (2021). For each password $p$, we selected the honeyword $h$ with highest cosine similarity as generated by FastText (Bojanowski et al., 2016), ensuring unique balanced pair per password. This strategy ensured that each password had a unique honeyword, allowing the CNN model to learn a balanced and optimized set of password-honeyword pairs with strong correlations.

**Data representation:** To facilitate the learning process of the CNN model, we employed character-level tokenization, treating each character in passwords and honeywords as an individual token. Due to the inherent variability in their lengths, we applied padding to standardize the character sequence lengths. This standardization is imperative for the training phase of the CNN model, which requires input data of consistent size.

### 5.2 Experiment Settings

We implemented the PassMine attack using two open-source libraries: `FastText` for training the representation learning model and `TensorFlow` for training and evaluating the CNN model.

**FastText training:** Following the implementation of Dionysiou et al. (2021) [2], we trained the FastText model on RockYou-large dataset. The choice of this dataset was motivated by its substantial size, which is critical for effective representation learning. A larger dataset provides more context and variation, enabling the model to generate high-quality honeywords that closely resemble real passwords. We used the default hyperparameters suggested by Dionysiou et al. (2021).

**CNN training:** We used the CNN model to distinguish passwords from honeywords using the base architecture described in Section 4. We employ neural architecture search (NAS) to automatically select effective configurations. Specifically, we use NAS to jointly optimize both architectural parameters (number of layers, filters, filter sizes, dropout rates) and training hyperparameters (optimizer type, learning rate, batch size) for each dataset–HGT combination. We ran NAS with a budget of 200 trials (or a 100-hour wall-clock cap), whichever comes first, per dataset–HGT pair, selecting the best configuration by validation loss. Once optimal parameters were identified, the 27 models (9 datasets $\times$ 3 HGTs) were trained with early stopping on validation loss using their respective configurations. The complete search space for the CNN models is reported in Appendix B.

For each dataset, we sample 45,000 password–honeyword pairs (i.e., 90,000 strings: 45,000 passwords and 45,000 corresponding honeywords) and split them into training (90%), validation (5%), and testing (5%). To assess generalizability to unseen accounts, we construct an independent evaluation set from the held-out 5,000 passwords (disjoint from the train/validation/test splits) from the original 50,000-password pool. From these 5,000 held-out passwords, we randomly sample 500 passwords for evaluation. For each sampled password, we generate a sweetword set of size $k$=20 (one real password and 19 honeywords), yielding 500 independent user accounts. This evaluation set is entirely separate from the data used in training, validation, and testing, ensuring an unbiased assessment.

### 5.3 PassMine Evaluation Metric: $\epsilon$-flatness

We evaluate the effectiveness of PassMine in identifying passwords within a limited number of attempts, comparing its performance against random guessing and Huang et al. (2024)'s attack.

The "$\epsilon$-flatness" metric measures the maximum probability that an adversary can correctly guess the real password from the set of sweetwords. Specifically, a system achieves $\epsilon$-flatness if the adversary's chance of correctly identifying the real password is no greater then $\epsilon$. For example, if $\epsilon$

[1] The code and dataset are available at: `https://bitbucket.org/srecgrp/honeygen-generating-honeywords-using-representation-learning/`. No license information is specified.

= 0.05, the adversary's probability of guessing the real password on the first attempt is limited to 5%.

In our evaluation of PassMine, we applied $\epsilon$-flatness metric to critically assess the efficacy of our password-guessing attack. However, the original metric proposed by Juels and Rivest (2013) does not account for scenarios where an attacker makes multiple guesses per user account, nor does it highlight system's most vulnerable sweetwords. To address these limitations, Wang et al. (2018) extended the $\epsilon$-flatness metric to accommodate multiple login attempts by an attacker, providing a more accurate representation of the effectiveness of HGTs in producing secure honeywords.

The extended $\epsilon$-flatness metric can be visualized using a flatness graph, where each point $(x, y)$ indicates that within the first $x$ attempts, the likelihood of correctly identifying the real password is $y$, for $x \leq k$. Here, $k$ represents the number of sweetwords per user account.

In our research, we consider the average $\epsilon$-flatness across multiple datasets for the group of 500 users to evaluate the overall performance of the attack. We designate the average $\epsilon$-flatness metric as success rate of PassMine in identifying passwords. This success rate reflects the attacker's probability of correctly guessing the real password within a specified number of attempts.

## 6 Evaluation & Results

We conducted two experiments: Experiment A (In-Distribution Evaluation), and Experiment B (Cross-Dataset Generalization) to evaluate PassMine.

**Experiment A. In-Distribution Evaluation.** In this experiment, we evaluate the effectiveness of PassMine when both the training and evaluation data are drawn from the same underlying distribution. For example, if the CNN model is trained on passwords and honeywords generated from the `Dubsmash.com` dataset, the evaluation is also performed on sweetwords from `Dubsmash.com` users (with 20 sweetwords per account). This setting reflects a scenario where an attacker's training data closely matches the target service and provides an upper bound on attack performance.

The average success rate across 500 users is markedly higher when compared to the baseline performance. As shown in Figure 3, PassMine substantially outperforms random guessing across all HGTs. For CbT and CbH, success rates exceed 50% on the first attempt, while CbP is more robust, still allows an attacker to succeed at nearly three times the baseline rate. Importantly, the success rate improves significantly as the number of login attempts increases. This highlights systematic vulnerabilities in the honeywords defense mechanism.

The flatness graphs for each HGT (Figures 3a, 3b, and 3c) further illustrates attack performance with increased login attempts. Notably, when using the CbP HGT, the performance curve remains slightly closer to the random guessing line compared to the performance curves observed with the CbT and CbH HGTs. This suggests that the latter two HGTs are more vulnerable to the PassMine.

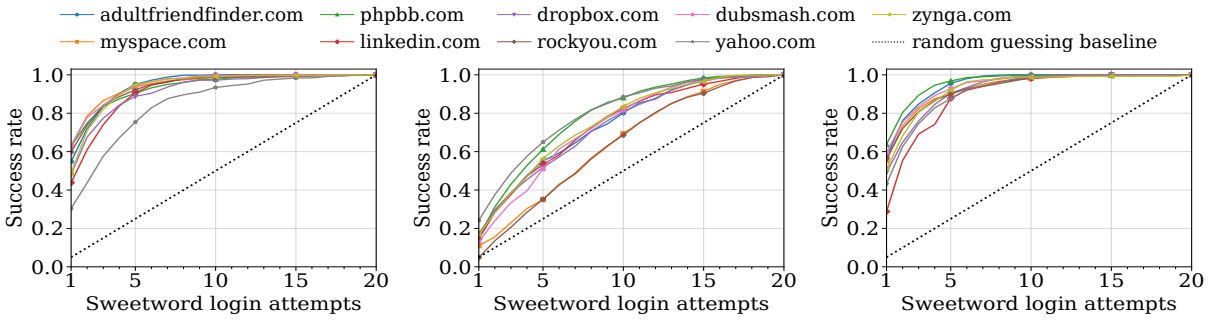
Figure 4 provides a summary of PassMine's performance as login attempts increase, reinforcing the observed trend of enhanced attack success with more login attempts.

**Experiment B. Cross-Dataset Generalization.** In this experiment, we evaluate cross-dataset generalization to different password distributions. We fix `Dubsmash.com` as the source dataset: we train (and tune all hyperparameters) using only `Dubsmash.com` passwords and honeywords, and then evaluate the resulting model *without any further tuning* on sweetwords constructed from each other target dataset (e.g., `MySpace.com`). This setting simulates an attacker who leverages leaked data from one web service to target users on another service, reflecting the impact of distribution shift caused by differing password policies or user populations.

PassMine consistently outperforms random guessing on the initial login attempt. However, the average success rate is slightly lower compared to the in-distribution settings. This discrepancy highlights the variability in password distributions across different services. Specifically, the success rates of the CbT, and CbP, and CbH HGTs are 51.84%, 8.22%, and 48.13%, respectively – each significantly exceeding the 5% baseline success rate of random guessing.

As the number of login attempts increases, the success rate improves substantially. For instance, after three login attempts, the success rates rise to 77.42%, 25.93%, and 75.53% for the respective HGTs. By the tenth login attempt, the rates further increase to 97.67%, 72.87%, and 97.76%, showcasing PassMine's strong password identification capabilities across services. These improvements are visualized in Figures 5a, 5b, and 5c.

Notably, the CbP HGT maintains a consistently




(a) CbT (b) CbP (c) CbH

Figure 3: Flatness graphs of PassMine in **Experiment A – In-Distribution Evaluation**. Each panel reports the success rate over sweetword login attempts for one HGT. Curves closer to the random guessing baseline indicate stronger security.

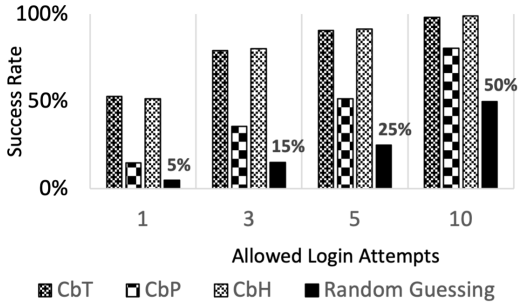



Figure 4: Summary of PassMine on different HGTs in **Experiment A - In-Distribution Evaluation** when number of allowed login attempts is 1, 3, 5, and 10.

low success rate on the first login attempt (though higher than random guessing), indicating its effectiveness in generating realistic honeywords. In contrast, the CbT and CbH HGTs demonstrate significantly higher success rates, allowing passwords to be identified with fewer attempts. The overall performance enhancement with increasing login attempts is summarized in Figure 6. By revealing vulnerabilities in existing HGTs, our work could enable more sophisticated attacks if misused.

**Comparison with Prior Work**: Figure 7 compares PassMine against the state-of-the-art attack by Huang et al. (2024) under cross-dataset generalization settings. Their approach ranks sweetwords using a contrastive DNN trained to produce password embeddings, scoring candidates by cosine similarity to cross-site auxiliary passwords from the target user. The model requires approximately 451 million training passwords and assumes attacker access to per-user cross-site data. We implemented their model using their code and evaluated it on our dataset. PassMine achieves higher success rates across all HGTs and attempt budgets. Within 3 attempts, it reaches ≈80% on CbT and CbH (vs. ≈15%) and ≈26% on CbP (vs. ≈12%). This gap is particularly notable given that PassMine uses a ≈10,023× smaller training corpus and requires no cross-site auxiliary data.

## 7 Discussions

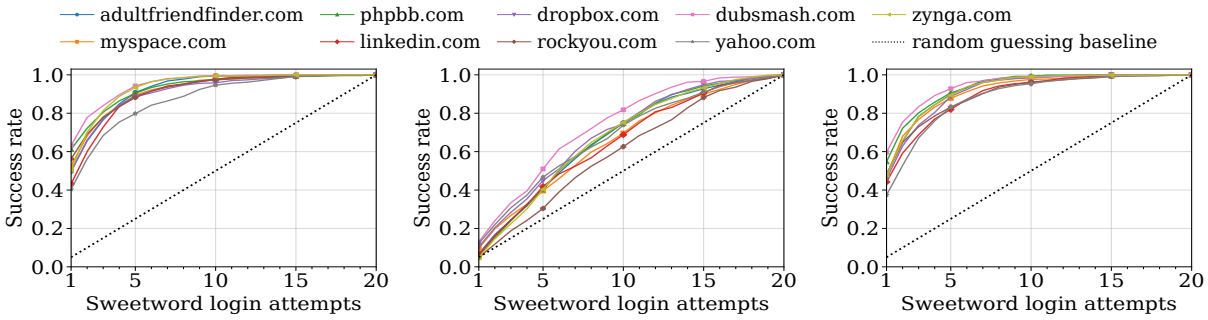
We discuss the performance of PassMine in exceptional cases, and mitigation strategies.

**Impact of Number of Sweetwords Per Account.** In our study, we studied the effectiveness of CbP and CbH, when more than 20 sweetwords are associated with the user account. We observed that PassMine can identify passwords with significantly higher success rate compared to random guessing. The CbT method is excluded from this analysis because prior experiments have already demonstrated its significant vulnerabilities, rendering it ineffective for practical security applications.

On the first login attempt, PassMine's average success rate is slightly higher than random guessing (3.73%). However, utilizing the CbH approach for honeyword generation elevates PassMine's success rate to 40.62%, significantly surpassing random guessing (3.33%). With increasing login attempts to 3, 5, or 10, the average success rate increases to 16.22%, 28.91%, or 54.55%, respectively, for CbP HGT. Similarly, for CbH, the average success rate increases to 66.28%, 79.33%, or 94.62% with 3, 5, or 10 login attempts, respectively. The results on individual datasets are presented in Table 5 of Appendix E.

These results demonstrate that PassMine remains highly effective even when more sweetwords are associated with each user account. The significant increase in success rates show that merely increasing the number of sweetwords does not deter attacks like PassMine, highlighting honeywords approach is fundamentally vulnerable.

**Impact of Numeric Passwords.** Some online passwords consist exclusively of numbers, often representing personal information such as a user's date




(a) CbT (b) CbP (c) CbH

Figure 5: Flatness graphs of PassMine in **Experiment A – In-Distribution Evaluation**. Each panel reports the success rate over sweetword login attempts for one HGT. Curves closer to the random guessing baseline indicate stronger security.

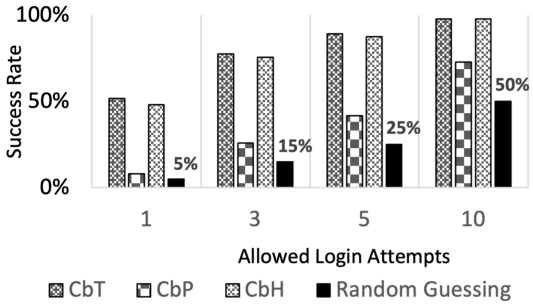



Figure 6: Summary of PassMine on different HGTs in **Experiment B - Cross Dataset Generalization** when number of allowed login attempts is 1, 3, 5, and 10.

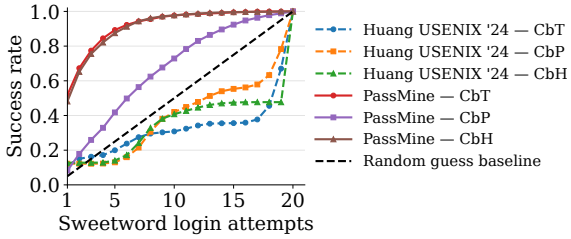



Figure 7: *Comparison of PassMine against the prior state-of-the-art attack by Huang et al. (2024).*

of birth, phone number, or area code. When generating honeywords for these numeric passwords, the resulting sequences are also entirely numeric, and the distributional gap between human-chosen and algorithmically-generated sequences is substantially smaller, leaving less exploitable signal for the classifier. Moreover, these honeywords can vary in length compared to the original numeric passwords, being either shorter or longer. Under these conditions, we observed a significant decline in the attack's performance. Specifically, using the CbP and CbH, the classifier achieved average success rates of only 0.8% and 1.2%, respectively, when 20 sweetwords were associated with each user account.

**Mitigation Strategies.** Our findings highlight the need for robust HGTs against DL-based attacks. First, future HGTs should minimize distributional divergence between real passwords and honeywords by training generative models against a discriminator. Second, honeyword generators should be adversarially hardened against discriminators such as PassMine, co-training the generator and discriminator so that generated decoys become progressively harder to distinguish. Third, deploying a randomized ensemble of HGTs per account can increase attacker uncertainty about the generating distribution, though our HGT-fingerprinting experiments suggest this alone may be insufficient (Appendix F). Finally, operational controls such as strict rate limiting and multi-factor authentication remain complementary defenses.

## 8 Conclusion

In this work, we introduced PassMine, a novel DL-based attack that uses CNNs to identify real passwords from a set of sweetwords. We also adapted the $\epsilon$-flatness metric for DL-based attacks, using model-predicted probabilities to rank sweetwords for password guessing. Our results show that PassMine can compromise the security of HGTs proposed by Dionysiou et al. (2021) without requiring any personally identifiable information. This suggests existing HGTs remain vulnerable and highlights the need for more robust HGTs. Our findings demonstrate that attackers equipped with DL models can undermine honeyword-based authentication and potentially delay the detection of data breaches. In addition, our framework can serve as a practical tool for benchmarking and evaluating future HGTs against DL-based attacks.

## Limitations

In this study, we focus on passwords and honeywords tied to user accounts, without considering

PII. However, future research could beneficially explore the impact of incorporating PII in identifying user passwords, and in the process of honeywords generation in a way that privacy of the user is preserved. Additionally, the use of passwords created by password managers (e.g. 1Password) or those generated by built-in browser managers (e.g., Chrome) can also be examined.

While our approach uses a character-level CNN for distinguishing passwords from honeywords, other DL architectures could be explored as well. For example, Long-Short Term Memory (LSTM), Bidirectional-LSTM (BiLSTM), and Transformers may be used to assess the efficacy of different HGTs or to improve the performance of the attack. In particular, set-level architectures such as a bidirectional Transformer operating over the entire sweetword set could explicitly model inter-candidate relationships, rather than scoring each candidate independently. Additionally, incorporating explainable AI (XAI) methods could provide greater insight into the features or patterns that the models are leveraging. This, in turn, could help guide the design of more robust defenses.

## Ethical Considerations

In accordance with the ethical framework proposed by Thomas et al. (2017), we used public datasets containing only passwords, ensuring there was no additional Personally Identifiable Information (PII), such as email addresses. The selection was imperative as our research relies on analyzing password datasets without causing undue harm or privacy breaches. Specifically, we ensured that the passwords under study could not be tracked to individual users. Our aim is not to identify users from these leaked passwords but to leverage them for generating honeywords and training DL models.

## References


Akshima, Donghoon Chang, Aarushi Goel, Sweta Mishra, and Somitra Kumar Sanadhya. 2019. Generation of secure and reliable honeywords, preventing false detection. *IEEE Transactions on Dependable and Secure Computing*, 16(5):757–769.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. 2010. Kamouflage: Loss-resistant password management. In *European symposium on research in computer security*, pages 286–302. Springer.

Nilesh Chakraborty and Samrat Mondal. 2015. A new storage optimized honeyword generation approach for enhancing security and usability. *CoRR*, abs/1509.06094.

Cybercrime Magazine. 2024. Who's hacked? latest data breaches and cyberattacks. Accessed: 2024-04-14.

Jimmy Dani and Boyang Wang. 2021. Hiddentext: Cross-trace website fingerprinting over encrypted traffic. In *2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI)*, pages 274–281.

Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. 2019. X-deepsca: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6.

Xavier de Carné de Carnavalet and Mohammad Mannan. 2014. From very weak to very strong: Analyzing password-strength meters. In *Network and Distributed System Security Symposium (NDSS 2014)*. Internet Society.

Antreas Dionysiou and Elias Athanasopoulos. 2022. Lethe: Practical data breach detection with zero persistent secret state. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 223–235.

Antreas Dionysiou, Vassilis Vassiliades, and Elias Athanasopoulos. 2021. Honeygen: Generating honeywords using representation learning. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 265–279, New York, NY, USA. Association for Computing Machinery.

Imran Erguler. 2016. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Dependable and Secure Computing*, 13(2):284–295.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander Rush. 2019. GLTR: Statistical detection and visualization of generated text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 111–116, Florence, Italy. Association for Computational Linguistics.

Yimin Guo, Zhenfeng Zhang, and Yajun Guo. 2021. Superword: A honeyword system for achieving higher security goals. *Computers & Security*, 103:101689.

Robert Hackett. 2021. Yahoo raises breach estimate to full 3 billion accounts.

Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Pérez-Cruz. 2017. Passgan: A deep learning approach for password guessing. *CoRR*, abs/1709.00440.

Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Perez-Cruz. 2019. Passgan: A deep learning approach for password guessing. In *International conference on applied cryptography and network security*, pages 217–237. Springer.

Baixiang Huang, Canyu Chen, and Kai Shu. 2025. Authorship attribution in the era of llms: Problems, methodologies, and challenges. *SIGKDD Explor. Newsl.*, 26(2):21–43.

Zonghao Huang, Lujo Bauer, and Michael K. Reiter. 2024. The impact of exposed passwords on honeyword efficacy. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 559–576, Philadelphia, PA. USENIX Association.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas Eck. 2020. Automatic detection of generated text is easiest when humans are fooled. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1808–1822, Online. Association for Computational Linguistics.

Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2021. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1143–1161. IEEE.

Ari Juels and Ronald L. Rivest. 2013. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &; Communications Security*, CCS '13, page 145–160, New York, NY, USA. Association for Computing Machinery.

Mina Esmail Zadeh Nojoo Kambar, Armin Esmaeilzadeh, Yoohwan Kim, and Kazem Taghva. 2022. A survey on mobile malware detection methods using machine learning. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0215–0221. IEEE.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature*, 521(7553):436–444.

Enze Liu, Amanda Nakanishi, Maximilian Golla, David Cash, and Blase Ur. 2019. Reasoning analytically about password-cracking software. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 380–397. IEEE.

Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. 2014a. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy*, pages 689–704. IEEE.

Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. 2014b. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy*, pages 689–704.

Microsoft. 2017. NNI: An open source AutoML toolkit for neural architecture search and hyper-parameter tuning.

Arvind Narayanan and Vitaly Shmatikov. 2005. 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. Association for Computing Machinery.

Fergus O'Sullivan. 2024. A timeline of dropbox security issues.

J Femila Roseline, GBSR Naidu, V Samuthira Pandi, S Alamelu alias Rajasree, and N Mageswari. 2022. Autonomous credit card fraud detection using machine learning approach⋆. *Computers and Electrical Engineering*, 102:108132.

Salted Hash-Top security news By Steve Ragan and Steve Ragan. 2016. Weebly data breach affects 43 million customers.

Jens Steube. 2023. Hashcat. `https://github.com/hashcat/hashcat`. Available at `https://github.com/hashcat/hashcat`.

Daniel R. Thomas, Sergio Pastrana, Alice Hutchings, Richard Clayton, and Alastair R. Beresford. 2017. Ethical issues in research using datasets of illicit origin. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 445–462, New York, NY, USA. Association for Computing Machinery.

Sharmila Kishor Wagh, Vinod K Pachghare, and Satish R Kolhe. 2013. Survey on intrusion detection system using machine learning techniques. *International Journal of Computer Applications*, 78(16).

Ding Wang, Haibo Cheng, Ping Wang, Jeff Yan, and Xinyi Huang. 2018. A security analysis of honeywords. In *NDSS*.

Ding Wang, Yunkai Zou, Qiying Dong, Yuanming Song, and Xinyi Huang. 2022a. How to attack and generate honeywords. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 966–983.

Ding Wang, Yunkai Zou, Qiying Dong, Yuanming Song, and Xinyi Huang. 2022b. How to attack and generate honeywords. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 966–983.

Ke Coby Wang and Michael K. Reiter. 2021. Using amnesia to detect credential database breaches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 839–855. USENIX Association.

Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. 2009a. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405.

Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. 2009b. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405.

Hans Weytjens and Jochen De Weerdt. 2020. *Process Outcome Prediction: CNN vs. LSTM (with Attention)*, page 321–333. Springer International Publishing.

Ming Xu, Jitao Yu, Xinyi Zhang, Chuanwang Wang, Shenghao Zhang, Haoqi Wu, and Weili Han. 2023. Improving real-world password guessing attacks via bi-directional transformers. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1001–1018.

## A Extended Literature Review

Juels and Rivest (2013) introduced the concept of honeywords to enhance the security of hashed passwords and suggested multiple techniques for their generation, including tweaking algorithms and model-based approaches (Bojinov et al., 2010; Weir et al., 2009b). They advocated for the use of 19 honeywords per account. Subsequently, Wang et al. (2018) applied these strategies to large datasets but found them inadequate, with attack success rates ranging from 29.29% to 67.9% across different scenarios. They developed an alternate method, utilizing PII to generate honeywords, reducing the success rate to 18.2% (Wang et al., 2022a).

Wang and Reiter (2021) proposed Amnesia, a technique that uses probabilistic marking to detect breaches without persistent state. Conversely, Dionysiou and Athanasopoulos (2022) noted limitations with Amnesia and proposed Lethe as an alternative, using honeywords to detect data breaches without needing a trusted component to distinguish passwords from user input credentials. Huang et al. (2024) critiques existing honeyword systems by proposing a threat model where attackers have access to legitimate passwords from other sites for the same users. This assumption potentially limits the applicability of its findings, as it does not account for less informed attackers who represent a more frequent threat.

Erguler (2016) developed 'honeyindex,' which employs other users' passwords as honeywords to minimize storage needs, though it faced deployment challenges (Wang et al., 2022a). Chakraborty and Mondal (2015) proposed Paired Distance Protocol (PDP), which enhances security by requiring users to remember a random string alongside their password, though it's susceptible to Multiple System Intersection attacks (Akshima et al., 2019). Akshima et al. (2019) studied advanced honeyword generation techniques (e.g., evolving-password-model, user-profile-model, append-secret-model) which prove resistant to certain attacks but vulnerable to others (Wang et al., 2022b). Guo et al. (2021) introduced Superwords, a method that disconnects usernames from passwords, significantly reducing the probability of successful attacks (Juels and Rivest, 2013).

## B Model Architecture Overview

The architecture of the CNN model employed for implementing PassMine is depicted in Figure 8. The model architecture initiates with an embedding layer, followed by repeated blocks that include convolution layers, batch normalization, activation functions, and pooling layers. The sequence culminates in a dense layer and dropout before making the final prediction. The CNN model predicts the likelihood of an input word being either a honeyword or a password, a fundamental aspect of PassMine's functionality. For comprehensive explanation of each component in this architecture, refer to Section 4.

To further enhance the performance of the CNN model, we conducted hyperparameter optimization using Microsoft (2017). Important hyperparameters, including learning rate, batch size, and model depth, were tuned using this framework. This automated approach ensures that the selected neural architecture is well-suited for the classification task and maximizes model accuracy and generalization. To mitigate overfitting, we applied dropout regularization and used early stopping based on validation loss. All hyperparameter selection was performed exclusively on validation splits, with the test set strictly held out and used only for reporting performance after hyperparameter selection. For the in-distribution experiments (Experiment A), we performed hyperparameter and architecture optimization separately for each dataset. For the cross-dataset generalization experiments (Experiment B), we used the model configuration tuned on the source dataset and evaluated it on the target dataset without further tuning. This design enables a fair assessment of both in-distribution performance and out-of-distribution generalization. The full hyperparameter search space is summarized in Table 1.

We conducted our experiments on a Linux workstation running Ubuntu 22.04.4 LTS, equipped with an Intel Xeon processor (3.6 GHz), 32 GB of RAM, and an NVIDIA RTX A5000 GPU.

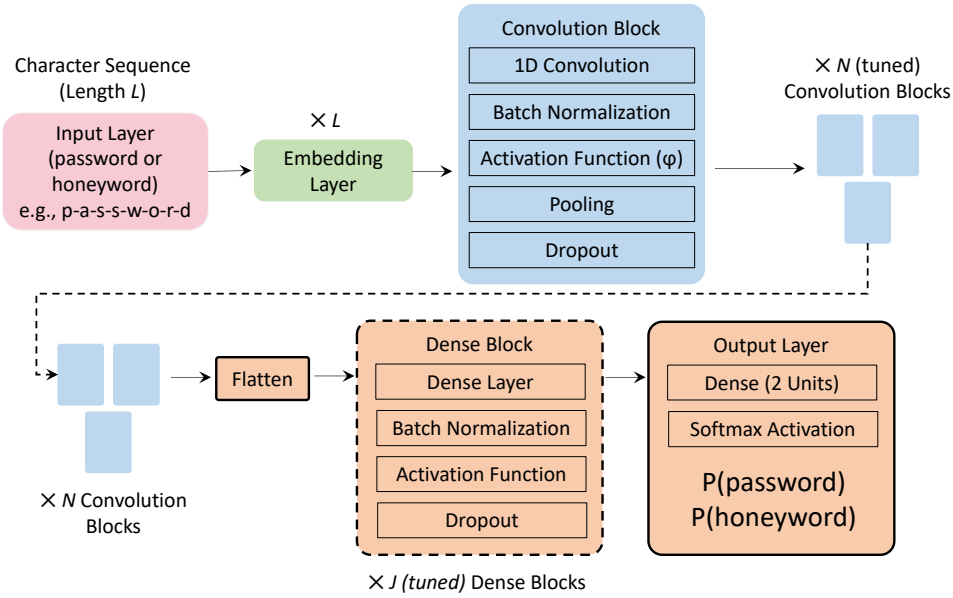



Figure 8: Character-level CNN architecture for binary password–honeyword classification. The model comprises N convolutional blocks for local feature extraction followed by J dense blocks. N and J are selected via Neural Architecture Search (NAS) for each dataset–HGT configuration.

| Hyperparameter | Search Space |
|---|---|
| Optimizer | RMSprop, Adadelta, Adagrad, Adam, Adamax, Nadam, SGD |
| Epochs | 400–3000 (stepwise choices) |
| Learning Rate | 0.00001 – 0.009 |
| Activation Functions | Softsign, ELU, Selu, ReLu, Tanh |
| Number of Conv Layers | 1, 2, 3, 4, 5 |
| Filter Numbers | 8, 16, 32, 64, 128, 256 |
| Kernel Size | 5, 7, 9, 11, 13, 15, 17, 19, 21 |
| Conv Stride Size | 2, 3, 4, 5, 6, 7, 8 |
| Dropout Rate (Conv) | 0.05, 0.1, 0.2, 0.3 |
| Number of FC Layers | 1, 2, 3, 4 |
| Embedding Dimension | 64, 128, 256, 512 |
| FC Activation Functions | Softsign, ELU, Selu, ReLu, Tanh |
| FC Dropout Rate | 0.05, 0.1, 0.2, 0.3, 0.4 |
| Batch Size | 8, 16, 32, 64, 128, 256 |
| Input Features | 70, 80, 90, 100 |

Table 1: Hyperparameter search space used for optimizing the 1D CNN character-level model. Each parameter was either tuned using choice-based or uniform sampling strategies. This includes architectural parameters (e.g., number of convolution and dense layers), training configurations (e.g., optimizer, learning rate), and activation functions for both convolutional and dense layers.

## C Dataset Details

We evaluate PassMine on nine publicly available password datasets provided by Dionysiou et al. (2021), sourced from compromised web services. Table 2 lists each dataset along with the number of passwords used. All datasets were preprocessed to exclude passwords shorter than eight characters, the RockYou dataset is provided in two versions (RockYou-small and RockYou-large) differing in size and sample selection criteria.

| Dataset | No. of Passwords |
|---|---|
| Adultfriendfinder | 50,000 |
| Dubsmash | 50,000 |
| Dropbox | 50,000 |
| Myspace | 50,000 |
| Phpbb | 50,000 |
| RockYou-small | 50,000 |
| LinkedIn | 50,000 |
| Yahoo | 50,000 |
| Zynga | 50,000 |
| RockYou-large | 1.04 million |

Table 2: Number of passwords available for each dataset.

| | chaffing-by-tweaking | | | | chaffing-with-a-password-model | | | | chaffing-with-a-hybrid-model | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ |
| **Adultfriendfinder** | 54.80% | 83.60% | 95.00% | 100.00% | 17.20% | 38.40% | 55.20% | 80.20% | 57.20% | 84.80% | 95.60% | 100.00% |
| **MySpace** | 63.00% | 86.60% | 93.60% | 98.80% | 11.00% | 23.00% | 35.00% | 69.40% | 55.40% | 82.40% | 90.00% | 98.20% |
| **phpBB** | 63.00% | 83.60% | 90.40% | 97.60% | 15.60% | 43.00% | 61.20% | 88.00% | 63.80% | 89.40% | 96.80% | 99.80% |
| **LinkedIn** | 43.80% | 74.00% | 90.60% | 99.40% | 14.60% | 37.60% | 53.40% | 82.40% | 28.80% | 69.20% | 87.80% | 98.20% |
| **Dropbox** | 48.20% | 77.40% | 88.80% | 97.20% | 16.20% | 38.60% | 51.80% | 80.80% | 49.20% | 76.00% | 90.40% | 99.20% |
| **RockYou-small** | 59.80% | 84.20% | 92.20% | 98.60% | 4.80% | 20.60% | 35.20% | 68.60% | 56.40% | 80.80% | 90.00% | 98.40% |
| **Dubsmash** | 63.20% | 84.00% | 94.20% | 99.60% | 13.00% | 33.40% | 51.00% | 81.80% | 59.60% | 83.40% | 92.80% | 98.80% |
| **Yahoo** | 30.40% | 57.60% | 75.40% | 93.40% | 24.20% | 48.80% | 65.00% | 88.60% | 43.20% | 74.60% | 87.60% | 99.00% |
| **Zynga** | 48.80% | 82.00% | 94.80% | 99.00% | 16.80% | 38.40% | 56.40% | 83.60% | 51.00% | 80.60% | 92.00% | 98.80% |
| **Average** | **52.78%** | **79.22%** | **90.56%** | **98.18%** | **14.82%** | **35.76%** | **51.58%** | **80.38%** | **51.62%** | **80.13%** | **91.44%** | **98.93%** |

Table 3: Average success rate of PassMine in **Experiment A (In-Distribution Evaluation)** in $1^{st}$, $3^{rd}$, $5^{th}$, and $10^{th}$ login attempt with different HGTs and 20 sweetwords per user account.

| | chaffing-by-tweaking | | | | chaffing-with-a-password-model | | | | chaffing-with-a-hybrid-model | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ |
| **Adultfriendfinder** | 49.80% | 76.60% | 90.80% | 99.40% | 6.80% | 24.40% | 40.80% | 75.00% | 48.20% | 78.40% | 89.80% | 99.20% |
| **MySpace** | 52.60% | 78.20% | 89.40% | 97.40% | 9.60% | 26.80% | 39.60% | 69.60% | 47.40% | 76.80% | 87.80% | 97.60% |
| **phpBB** | 61.20% | 80.60% | 90.40% | 97.80% | 6.00% | 24.00% | 40.20% | 74.80% | 54.80% | 80.20% | 90.60% | 99.20% |
| **LinkedIn** | 43.20% | 72.80% | 88.80% | 97.60% | 6.20% | 24.40% | 41.80% | 68.80% | 44.20% | 68.80% | 82.00% | 95.80% |
| **Dropbox** | 49.60% | 77.00% | 88.60% | 95.80% | 10.40% | 28.40% | 45.00% | 74.20% | 45.80% | 73.80% | 88.80% | 98.40% |
| **RockYou-small** | 56.20% | 77.80% | 88.20% | 97.40% | 5.00% | 18.80% | 30.40% | 62.60% | 48.00% | 73.00% | 83.20% | 96.40% |
| **Dubsmash** | 63.20% | 84.00% | 94.20% | 99.60% | 13.00% | 33.40% | 51.00% | 81.80% | 59.60% | 83.40% | 92.80% | 98.80% |
| **Yahoo** | 40.40% | 68.40% | 79.80% | 94.60% | 12.20% | 30.80% | 46.80% | 73.80% | 37.40% | 67.20% | 83.20% | 95.40% |
| **Zynga** | 50.40% | 81.40% | 93.40% | 99.40% | 4.80% | 22.40% | 40.40% | 75.20% | 47.80% | 78.20% | 89.40% | 99.00% |
| **Average** | **51.84%** | **77.42%** | **89.29%** | **97.67%** | **8.22%** | **25.93%** | **41.78%** | **72.87%** | **48.13%** | **75.53%** | **87.51%** | **97.76%** |

Table 4: Average success rate of PassMine in **Experiment B (Cross-Dataset Generalization)** in $1^{st}$, $3^{rd}$, $5^{th}$, and $10^{th}$ login attempt with different HGTs and 20 sweetwords per user account.

## D Results of PassMine on individual datasets

In this section, we provide supplementary material for experiments conducted for each of the threat models. The results presents the $\epsilon$-flatness (success-rate) of PassMine when attacker is allowed more than 1 attempt to log in to the system. Table 3, and 4 shows the average success rate of PassMine for password identification when 20 sweetwords are associated with the user account when attacker make 1, 3, 5, or 10 login attempts.

## E Effectiveness of PassMine with Increased Sweetwords Per Account

In this section, we provide additional details on how increasing number of sweetwords per user account affects the effectiveness of PassMine.

Table 5 demonstrates the effectiveness of the proposed PassMine attack across multiple datasets when honeywords are generated using two different approaches: CbP and CbH. The results are presented in settings when an attacker makes the $1^{st}$, $3^{rd}$, $5^{th}$, and $10^{th}$ login attempts, reflecting the success rate of PassMine in identifying passwords with increasing login attempts.

These findings affirm the discussions presented in Section 7 where it is noted that PassMine significantly outperforms random guessing and remains effective even as the number of sweetwords increases. The persistent high success rates across different HGTs suggest that while it offers a better defense than the password-only model, the overall approach requires further refinement to effectively counter against powerful DL-based attacks, such as PassMine. Additionally, our study also emphasized the need for continuous improvements in honeywords systems and suggests a reevaluation of how complexity in security models might inadvertently aid attackers rather than deter them.

## F The HGT Fingerprinting Experiment

As discussed in Section 7, an attacker operating in a black-box setting can use HGT fingerprinting to identify the generation technique used by the target system. Specifically, the attacker trains a lightweight classifier on honeywords generated by each known HGT, using aggregated distributional statistics as features. Given an observed honeyword set from the target, the attacker applies this classifier to identify the most likely HGT before deploying PassMine. To evaluate this strategy, we trained such a classifier across the HGTs considered in this work and obtained a top-2 accuracy of 91%, indicating that in most cases the attacker can

| | chaffing-with-a-password-model | | | | chaffing-with-a-hybrid-model | | | |
|---|---|---|---|---|---|---|---|---|
| **Dataset** | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ | $1^{st}$ | $3^{rd}$ | $5^{th}$ | $10^{th}$ |
| **Adultfriendfinder** | 3.80% | 14.00% | 25.80% | 52.40% | 42.20% | 66.00% | 80.40% | 96.40% |
| **MySpace** | 3.80% | 17.00% | 29.80% | 51.60% | 40.80% | 69.60% | 82.20% | 96.00% |
| **phpBB** | 3.60% | 14.00% | 25.80% | 55.20% | 46.80% | 71.60% | 81.60% | 96.00% |
| **LinkedIn** | 2.00% | 16.00% | 28.80% | 52.00% | 35.40% | 61.20% | 75.40% | 92.20% |
| **Dropbox** | 5.20% | 18.80% | 32.00% | 58.40% | 37.00% | 63.60% | 77.80% | 93.80% |
| **RockYou-small** | 2.20% | 12.00% | 20.80% | 43.60% | 39.40% | 64.20% | 77.60% | 91.60% |
| **Dubsmash** | 4.40% | 21.20% | 36.40% | 65.00% | 53.20% | 73.80% | 85.00% | 97.40% |
| **Yahoo** | 6.80% | 20.20% | 34.80% | 56.40% | 31.00% | 56.00% | 71.80% | 91.40% |
| **Zynga** | 1.80% | 12.80% | 26.00% | 56.40% | 39.80% | 70.60% | 82.20% | 96.80% |
| **Average** | **3.73%** | **16.22%** | **28.91%** | **54.56%** | **40.62%** | **66.29%** | **79.33%** | **94.62%** |

Table 5: Average success rate of PassMine in $1^{st}$, $3^{rd}$, $5^{th}$, and $10^{th}$ login attempt with different HGTs and 30 sweetwords per user account.

correctly narrow down the target HGT to a small set of likely candidates.