ABSTRACT
One approach to measuring password strength is to assess the probability it will be cracked in a fixed set of guesses. The current state of the art in password guessing employs a first-order Markov model that makes several assumptions about the distribution of passwords. We present two novel approaches to modeling password distributions that remove some of these assumptions. First, a layered Markov model is developed that extends a first-order model with index-sensitive weights. This model learns a separate set of transition probabilities for every $i$th character in a password. We validate our approach on a collection of publicly available datasets of real-world passwords. Our results indicate that the layered model increases performance by as much as 157%. Second, we present a technique that, given an approximate model, learns a better distribution of passwords on-line, while simultaneously cracking passwords. Our approach also has the appealing property that the number of guesses required to learn is inversely proportional to the number of passwords in the database. We demonstrate the utility of our algorithm with a realistic example of two distinct distributions and show that the model’s performance increases by 143% after the learning phase.

Categories and Subject Descriptors

General Terms
Passwords, Markov Models, Evolutionary Algorithms, Cryptanalysis

1. INTRODUCTION
As tools and services continue to shift into the cloud, users are increasingly finding their data stored in remote databases with various levels of security. Access to these accounts are typically protected by a user-selected password that is then stored in the database. In the event of a database being compromised by an attacker, a user’s password is only secure if the database was properly hashed and salted, and if the user chose a password that is difficult for the attacker to guess. While modern web application authentication frameworks, such as AuthLogic for Ruby on Rails [1] and Forms Authentication for ASP.NET MVC [2], typically include hashing and salting of passwords, many organizations continue to rely on older, less-secure systems that may only hash passwords or even store them in plaintext. Correspondingly, recent years have seen a rise in attacks on web application databases, resulting in a number of large databases of real-world passwords being made available online [4].

Even if a compromised database has been properly encrypted, the passwords are still vulnerable if they are chosen according to a predictable pattern or from a small distribution of possible passwords such as a dictionary of words. Indeed, studies show that users frequently choose low-entropy passwords that contain only lowercase letters [6]. As a countermeasure to this, applications generally enforce a set of constraints that user passwords must satisfy. Often such constraints are encoded in simple rules such as “must contain at least one number” or “must be at least 8 characters long.” These constraints are designed to force a user to increase the entropy of their password, thereby making it more difficult to crack.

While the intention of rule sets are to force users to choose more random passwords, human users have consistently chosen to simply adapt their predictable patterns to match minimum requirements of new rule sets [13, 5, 6]. For instance, a user who wants to enter “password” as their password may be thwarted by a rule requiring a number is present. Rather than selecting a new password at random that satisfies the rule, the human user is more likely to make a small edit to their initial choice, such as using “password1” as their password. Although this change is valid according to the rule set, it is nonetheless not substantially more secure than the original password because the user is still following a predictable pattern.

These predictable patterns have been exploited effectively by multiple techniques. Modern dictionary attacks, such as those used by the cracking tool John the Ripper (JtR) [3], have adapted to this by including a mangling phase that perturbs the dictionary according to a set of hand-crafted rules that mimic how a human is likely to perturb their password. Probabilistic context-free grammars can be crafted to generate better guesses based on commonly observed password syntax [14] and can easily be reused to create implicit creation rules [13]. Alternatively, Markov models can be used to order guesses based on the probability of each sequence of characters being typed [10]. If the database is only hashed and individual records are not salted, then the hashed records are vulnerable to rainbow table attacks [11] that can be enhanced with a hybrid Markov model.
While highly effective compared to brute force methods, the Markov models described in [10] and implemented in JtR rely on two key assumptions. First, all characters in a password are assumed to be identically distributed after the initial character and any additional information about the distribution of characters at each index of the password is discarded. Second, the distribution of passwords and character transition probabilities in the training set is assumed to be representative of passwords in any database.

In practice, neither of these assumptions should hold true. The distribution of passwords is strongly affected by the creation rules applied to the database and such rules vary across organizations, with some placing no constraints on passwords, others enforcing explicit rules about minimum lengths and different character types, and recent implicit approaches [13] based on likelihood of a password being guessed. Every new iteration of creation rules fundamentally alters the distribution of user-selected passwords in ways that may be unknown to the attacker, but are no doubt still driven by a predictable generative pattern optimized more for easy human recall than difficulty of cracking. Thus, improving the capability of Markov models to accurately represent known password distributions and to adapt to new, unknown distributions will enable us to better understand the objective strength of a specific password and may lead to a more precise measurement of guessability.

To this end, we present two novel approaches to modeling password distributions that remove the assumptions described above. First, a layered Markov model is developed that extends a first-order model with index-sensitive weights. This model learns a separate set of transition probabilities for every \( i^{th} \) character in a password. We validate our approach on a collection of publicly available datasets of real-world passwords. Our results indicate that the index-aware model can increase accuracy by as much as 153\% over the model currently used in the popular cracking tool John the Ripper.

If a different set of creation rules were enforced between the training and target databases, they may have significantly different distributions of passwords. To handle this scenario, we present a technique that, given an approximate model such as one trained on a database with no creation rules, learns a better distribution of passwords on-line, while simultaneously cracking passwords. Our approach also has the appealing property that the number of guesses required to learn is inversely proportional to the number of passwords in the database. We demonstrate with a realistic example of training a model on a database with no creation rules and then evolving it on-line against an encrypted database with creation rules requiring passwords to be at least eight characters and contain at least one number. Our results indicate that the model’s performance increases by 143\% after the learning phase.

This paper makes the following novel contributions:

- An improved Markov model capable of more accurately representing a distribution of passwords.
- A comparative analysis of the current state-of-the-art Markov model with our improved model on several real-world datasets.
- A machine learning algorithm that takes a model trained on a similar distribution and can improve it on-line, while simultaneously guessing passwords, to better model the target distribution.

The rest of this paper is organized as follows. Section 2 presents background information and related work. Section 3 describes our improved Markov model. Section 4 presents our machine learning algorithm for improving a model on-line. Section 5 presents empirical results of our improved model on several real-world datasets and the results of our learning algorithm on a realistic example. In Section 6 we discuss insights gained from our analysis and outline future work. Finally, Section 7 presents our conclusions.

2. BACKGROUND

We next discuss relevant background work related to the state of the art in using Markov models for password cracking and the machine learning technique we use in our on-line learning algorithm.

2.1 First-Order Markov Models

Currently, the most advanced model, first presented in [10] and later implemented in JtR, is a first-order Markov chain as shown in Figure 1a. The model generates passwords probabilistically according to the weighted, directed edges between nodes. Sampling a password begins at the origin, where each edge stochastically transitions to one of the 95 nodes in the main layer, each corresponding to a different ASCII password character. The nodes in the main layer then transition between themselves in an analogous manner. At each transition, the character for that node is appended to the guess and the model is iterated for a fixed password length. As the model contains only an origin node and a single layer of character nodes, it is called a first-order Markov model.

To create a password cracking model, one first trains the Markov model over a known distribution. This distribution can simply be a publicly available set of passwords from a previously attacked dataset. For instance, JtR uses the rockyou [4] dataset for its Markov model. For efficiency purposes, the Markov model can be converted from a probabilistic graphical structure to an ordered mapping based on the probability of each path, in descending order. This enables the model to be used to structure the order of guesses and can also be applied to rainbow table attacks [10].

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [7] are a policy search method based on the process of Darwinian evolution. A population of individuals is evaluated based on a user-defined fitness metric. After every individual has been evaluated, natural selection is performed on the population, with the fittest individuals surviving and creating randomly mutated offspring. This process is repeated either for a fixed number of steps or until a predefined population size is reached.

Valid password characters are assumed to be in the range of [32, 126], but our approaches are trivially extensible to other character sets.
generations or until some other stopping condition, such as a global optima, is found. As the original application of EAs was to learn finite state machines, they seem well-suited to evolving Markov models.

Specific to our implementation, we modify the NEAT algorithm [12] to support Markov model evolution. NEAT was originally created to evolve artificial neural networks, but has features useful for evolving any graphical structure, such as support for arbitrary topology, speciation by graphical similarity, and fitness sharing to support innovation and encourage diversity in the population.

In the next section, we present our new representation that builds on the first-order Markov model concept.

3. LAYERED MARKOV MODEL

When creating a first-order Markov model, much information is lost about the true generative model underlying the distribution of passwords. For instance, while the model is able to differentiate the first character in a word from all later characters (via the first set of transition probabilities from the origin node), the remaining characters are then treated as being identically distributed. Clearly this assumption is incorrect, since we know that when users add numbers to their alphabetic passphrases they overwhelmingly choose to append them to the end [13]. It seems intuitive that other index-dependent password rules may also exist.

To capture such rules, we extend the first-order Markov model into an \( n \)-layered model, where \( n \) is the length of the desired password(s) to crack. Figure 1b shows an example of our password model with \( n = 4 \). The model is composed of a set of layers that are trained with index-sensitive weights up to the \( n^{th} \) layer, at which point all remaining probabilities are grouped together.

It is worth noting that we have intentionally kept this model compact compared to other options. For instance, we could have created an \( n^{th} \)-order Markov model in which each node transitions to its own set of nodes, effectively capturing both index and character information about all previous characters in the current state. However, such an approach would cause a prohibitively large expansion in the state space, likely causing the model to be slow and requiring an unrealistic number of samples for adequate training. Additionally, converting the graph into a deterministic ordering for efficient guessing, as in [10], would be infeasible as the number of paths would be too large. We therefore believe this layered approach achieves an important balance between granularity and combinatorial explosion.

This layered representation enables us to capture more information about a known distribution of passwords. In the next section, we present our approach for learning a model when the distribution is unknown.

4. ON-LINE LEARNING ALGORITHM

Most leaked or hacked password databases are heavily skewed towards weak passwords composed mostly of lowercase letters. This is especially true of the largest hacks, such as those analyzed in [13], which all enforced no password creation rules at all. This is most likely because the systems that are most vulnerable to attack are those which are outdated and fail to follow the common practice among application developers. Rule sets employed at creation time, whether known or unknown to the attacker, also mean that the distribution of passwords may be specific to the database itself. Intuitively, training on a different database (e.g., the rockyou database that JtR relies on) may even be detrimental to performance if the set of creation rules for the target database used a similar model internally to reject predictable passwords [14, 13].

Mainstream adoption of creation rules is also constantly evolving. Common practice has moved from no requirements, to including one uppercase and number, to now fre-
quently also requiring a special character. In the extreme case, one may not even know the rule set and may be faced with a target database of hashed and salted passwords of unknown distribution. In such a case, one is left with three options: 1) perform a brute force search under the worst-case assumption that the password rules forced users to choose truly random passwords, 2) use some combination of known password distributions to train a model in the hopes that the unknown distribution is sufficiently similar, or 3) attempt to approximate the unknown distribution by performing a search over the space of possible models. To the best of our knowledge, all previous work has focused on one of the first two options; our approach addresses the third option.

We cast the password cracking challenge as a reinforcement learning problem and learn, via an EA, a policy for generating password guesses. At each step the agent guesses a password and receives a reward equivalent to the number of accounts that match the guess, excluding duplicates of previous guesses by that agent. Each guess is generated by stochastically traversing the Markov model; while this produces some guesses that have already been found in the database, they are necessary and even desirable as we wish to compare models of the entire database, not just the portion that remains encrypted. The fitness score for an individual is equivalent to the sum of all the rewards it received from its guesses. The population of individuals is then evolved using a variant of the NEAT algorithm adapted for Markov models.

Our approach has several key benefits for password cracking. First, EAs are embarrassingly parallel as each individual can be evaluated independently on a separate process. Our algorithm also operates on-line, learning the distribution while simultaneously cracking passwords, reducing the burden of the learning process. Since the number of valid guesses defines the granularity of the search space, increasing the number of passwords to crack actually smoothes the fitness landscape and enables the algorithm to make more incremental progress. Thus, our approach also has the appealing property of scaling inversely proportional to the size of the database.

Next we describe the experimental setup and results that validate our \( n \)-layered Markov model and our learning algorithm.

5. EXPERIMENTS AND RESULTS

We conducted separate sets of experiments to evaluate our two approaches. To validate our layered Markov model, we ran it against a collection of real-world password datasets and compared its results to that of a first-order model. For our EA, we created two realistic datasets representing fundamentally different password distributions and showed that a simple first-order model trained on the first database can be drastically improved by evolving it on-line against the second database.

5.1 First-Order vs. 8-Layers

To evaluate the effectiveness of our \( n \)-layered model, we compared its performance to a first-order model on five real-world datasets: faithwriters, singles.org, phpbb, rockyou, and myspace. The first four have previously been analyzed in-depth and the last is a database of social networking passwords acquired via a phishing scam. All password datasets were acquired by unaffiliated third parties and are publicly available online. In the case of the myspace dataset, the real database had creation rules in place that required a minimum of one numeric character; however, the phishing website simply accepted any input from the user and thus only 85% of passwords in the raw dataset contain a number. We filtered out the 15% of passwords that could not be in the actual database and refer to the resulting cleaned dataset as the myspace dataset. A summary of the distributions of passwords in each database is presented in Table 1.

For each of the five databases, both a first-order and an 8-layered model were trained on the distribution of passwords in that database. Each trained model was subsequently run for \( 3 \times 10^6 \) guesses on the four other databases and the training database, for a total of 25 runs per model. Figure 2 shows a comparison of the performance of the first-order and 8-layered models on each database. The layered model clearly dominates the first-order model across all databases, outperforming the first-order model in all 25 tests by as much as 157.65%. Figure 2f shows the average performance gain of the layered model over the first-order model across each testing database.

![Table 2: Summary statistics for the realistic example databases. Both datasets are created from a perturbed English language dictionary. The right four columns show the percentage that contain each type of character: L = lowercase, U = uppercase, N = number, and O = other character.](image)

### Table 2: Summary statistics for the realistic example databases. Both datasets are created from a perturbed English language dictionary. The right four columns show the percentage that contain each type of character: L = lowercase, U = uppercase, N = number, and O = other character.

<table>
<thead>
<tr>
<th>Database</th>
<th>Passwords</th>
<th>L</th>
<th>U</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>training</td>
<td>354296</td>
<td>100%</td>
<td>0%</td>
<td>9.94%</td>
<td>1.71%</td>
</tr>
<tr>
<td>testing</td>
<td>354296</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
<td>1.71%</td>
</tr>
</tbody>
</table>

5.2 Learning an Unknown Distribution

Aside from the filtered myspace dataset, all of the real-world password databases contain no creation rules and are consequently composed of highly similar password distributions. Thus, to validate our EA we have chosen to create two realistic example databases, each a perturbed version of an English dictionary of 354K words.

In our training dataset, we randomly mutate 10% of the dictionary entries to contain a number; though this distribution does not match any of the benchmark datasets used in the last experiment, it is not unrealistic as a recent study showed that more than 90% of seven-character passwords contain only lowercase letters. The testing dataset simulates the same dictionary with creation rules enforced that require a password to be at least eight characters long and contain at least one number; for each entry in the dictionary that is less than seven characters, the length constraint is satisfied by adding multiple digits. In both datasets, the

\[3\text{We are currently running an evolution on the myspace dataset, however due to its small size the run required a large number of guesses per individual and was not able to finish by the paper deadline.}\]
digit insertions are sampled from a non-uniform distribution similar to the pattern found in the rockyou database [13]. Table 2 summarizes the distribution of passwords in each of our example databases.

To evaluate the performance of our on-line learning algorithm, we first train a first-order Markov model on the training dataset. We intentionally chose the simpler first-order model instead of a layered model so as to make the comparison be against the currently-accepted state of the art. This model is run for $10^8$ guesses and serves as the baseline for comparison to our EA. The baseline model is then used as a seed for a population of 100 individuals, each evaluated on $10^8$ guesses per individual. The algorithm is run for 200 generations, producing $10^7$ guesses per generation.

After 200 generations of evolution, the resulting model produces nearly 143% more correct passwords in $10^8$ guesses than the baseline. Furthermore, as shown in Figure 3, by generation four, after only $4 \times 10^7$ guesses, the on-line evolutionary algorithm surpasses the the number of correct guesses discovered by the baseline model in $10^8$ guesses. These results suggest that the algorithm is not only effective at learning the distribution, but also at speeding up password cracking while still in the process of learning.

In the next section, we present a brief discussion of our results and outline future work.

Table 1: Summary statistics for the five real-world password databases used to validate our approach. The first four databases all contained no creation rules and thus have similar distributions. The myspace database had a creation rule requiring passwords contain at least one number, effectively skewing the distribution of passwords.

<table>
<thead>
<tr>
<th>Database</th>
<th>faithwriters</th>
<th>singles.org</th>
<th>phpbb</th>
<th>rockyou</th>
<th>myspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Passwords</td>
<td>8347</td>
<td>12233</td>
<td>184389</td>
<td>14344388</td>
<td>31326</td>
</tr>
<tr>
<td>Contains Lowercase</td>
<td>91.55%</td>
<td>88.54%</td>
<td>86.46%</td>
<td>78.45%</td>
<td>95.82%</td>
</tr>
<tr>
<td>Contains Uppercase</td>
<td>9.94%</td>
<td>9.84%</td>
<td>10.10%</td>
<td>9.32%</td>
<td>6.62%</td>
</tr>
<tr>
<td>Contains Number</td>
<td>44.55%</td>
<td>38.11%</td>
<td>54.11%</td>
<td>68.08%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Contains Non-alphanumeric</td>
<td>0.55%</td>
<td>0.25%</td>
<td>2.21%</td>
<td>7.10%</td>
<td>3.29%</td>
</tr>
</tbody>
</table>

6. DISCUSSION AND FUTURE WORK

Both our layered model and learning algorithm have shown substantial performance improvement over the current approach. In this section we discuss the insights we gained from our analysis and describe potential future work.

Our current approach relies on stochastic activation to generate each guess for a model. While this method is likely a good approximation of the effectiveness of a Markov model, in practice these models are converted into precomputed tables based on the probability of each possible path. Large models pose the risk of a combinatorial explosion in path counts and may make the model infeasible to implement in tools such as JtR. Similarly, large lookup tables may not fit well into the processor’s cache and thus suffer a performance penalty that negates their accuracy gains over more compact models. We believe that our layered model fits well into these constraints, but plan to demonstrate this concretely by implementing our 8-layer model in JtR and running an in-depth comparison of to its current model.

Conceptually, it is clear that creation rules affect the distribution of passwords in a database and that a variety of such policies are in use across different organizations. However, none of the larger databases that have been made publicly available, such as rockyou or phpbb, contained any creation rules. Although the myspace database did enforce (after filtering) a creation rule requiring at least one number, the relatively small size of the database means that our learning algorithm requires a large number of guesses to improve its model. Thus, we have started an experiment to learn a better first-order model by training on phpbb and testing on myspace. It is expected to take around 200 hours to complete and the results are consequently still pending.

In addition to the biases our work seeks to address, other biases are frequently introduced into password guessing models. Most attackers follow an increasingly aggressive plan of attack, starting with a dictionary of common passwords, then mangling that dictionary to include numbers and non-alphanumeric characters, and then finally relying on a Markov model to intelligently brute-force the attack. Consequently, Markov models, such as the one in JtR, that train on an entire database like rockyou are learning a distribution over all passwords rather than all important passwords. A better approach may be to first filter out passwords that an attacker would find by other means and then training on the resulting difficult passwords.
In a similar manner, the assumption that all passwords in a database were drawn from the same distribution and can be best modeled with a single Markov model seems naive. Rather, it is more likely that a database contains a mixture of samplings from different, possibly even disjoint, distributions. It may therefore be possible to improve performance by first clustering passwords into different categories and then training a collection of models or training models using a technique like boosting [8] that prunes out the passwords a model can already generate easily.

Finally, we plan to explore ways to enhance our learning algorithm. It may be possible to achieve better results by replacing the NEAT algorithm with an Estimation of Distribution Algorithm (EDA) [9] that explicitly tries to model the distribution of solutions in place of crossover and mutation. We also plan to investigate combining our layered model and learning algorithm.

7. CONCLUSION
As modern password databases are often hashed and salted, more intelligent guessing is the primary avenue for increasing password cracking rates. Thus, better modeling of password predictability is crucial to increasing the precision of password strength estimates. We have presented two novel approaches that improve on the current state-of-the-art models for predicting passwords. We demonstrated that a layered, index-sensitive Markov model outperforms a first-order Markov model on a collection of real-world password datasets. We also showed that when the training and target databases are drawn from different distributions of passwords that it is not only possible but actually beneficial to learn a better model of the target database on-line, while simultaneously making guesses. We believe these two contributions represent significant improvements compared to existing techniques and their success reveals a range of new possibilities for research in password strength analysis.

8. REFERENCES
Figure 2: Performance results comparison of a first-order Markov model to our 8-layer Markov model on the a) faithwriters, b) myspace, c) phpbb, d) rockyou, and e) singles.org datasets, as well as f) overall average improvement per dataset using our approach.