The Security of Modern Password Expiration: An Algorithmic Framework and Empirical Analysis

Yinqian Zhang
University of North Carolina at Chapel Hill
Chapel Hill, NC
yinqian@cs.unc.edu

Fabian Monrose
University of North Carolina at Chapel Hill
Chapel Hill, NC
fabian@cs.unc.edu

Michael K. Reiter
University of North Carolina at Chapel Hill
Chapel Hill, NC
reiter@cs.unc.edu

ABSTRACT

This paper presents the first large-scale study of the success of password expiration in meeting its intended purpose, namely revoking access to an account by an attacker who has captured the account's password. Using a dataset of over 7700 accounts, we assess the extent to which passwords that users choose to replace expired ones pose an obstacle to the attacker's continued access. We develop a framework by which an attacker can search for a user's new password from an old one, and design an efficient algorithm to build an approximately optimal search strategy. We then use this strategy to measure the difficulty of breaking newly chosen passwords from old ones. We believe our study calls into question the merit of continuing the practice of password expiration.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection—*Authentication*; H.1.2 [**MODELS AND PRINCIPLES**]: User/Machine Systems—*Human factors*

General Terms

Security, Human Factors

Keywords

User authentication, passwords, password expiration

1. INTRODUCTION

The practice of regularly expiring passwords has been a staple of computer security administration for over a quarter century (e.g., [5]). With few exceptions (e.g., [24, 3]), this practice is nearly universally accepted as a basic tenet by which systems should be protected, the common wisdom being:

> *Changing passwords frequently narrows the window within which an account is usable to an attacker before he has to take additional steps to maintain access. ... Password expiration does not offer any benefit when an attacker wants to do all of the damage that he's going to do right now. It does offer a benefit when the attacker intends to continue accessing a system for an extended period of time.* [2]

At this level of specificity, such an argument is unquestionably sound. However, the process of reducing such intuition to a reasonable password expiration policy would ideally be grounded in measurements of what "additional steps" the policy hoists on an attacker, so as to be certain that these "additional steps" are an impediment to his continued access. Unfortunately, even to this day, the security community has yet to provide any such measurements.

In this paper we provide the first analysis of which we are aware of the effectiveness of expiring passwords. Using a dataset of password histories for over 7700 defunct accounts at our institution, we assess the success with which an attacker with access to one password for an account can break a future password for that account, in either an offline fashion where the attacker can test many password guesses or an online one where the attacker is limited to only a few. Central to our analysis is the development of a *transform-based* algorithmic framework that an attacker can employ for breaking future passwords given preceding ones. Transform-based algorithms build from the presumption that a typical user will generate her next password by making systematic modifications to her current one (i.e., by applying primitive *transforms*).

The conjecture that users tend to generate future passwords based on old passwords is by no means new. The best evidence we have found in the literature to support this conjecture is a study of password systems reported by Adams and Sasse [1], comprising 139 responses to a web-based questionnaire and 30 semi-structured in-depth interviews. The hazard of primary concern in this paper was documented there as follows:

> *Some users devise their own methods for creating memorable multiple passwords through related passwords (linking their passwords via some common element) — 50% of questionnaire respondents employed this method. Many users try to comply with security rules by varying elements in these linked passwords (name1, name2, name3, and so forth).*

Although Adams and Sasse reveal that 50% of questionnaire respondents reported "linking their passwords via some comment element", it is left unresolved as to whether these linkages are typically of such a trivial variety. After all, many semantic linkages (e.g., passwords developed from the first names of the members of a family with which the user is acquainted) may not be nearly so simple to exploit in an automated fashion, while still representing "related passwords" to the user. Quantifying the pervasiveness of

CCS'10, October 4–8, 2010, Chicago, Illinois, USA.

easily exploited linkages between old and new passwords is at the heart of what we explore in this paper.

Specifically, we consider the challenge of attacking future passwords from past ones for the same accounts using a transform-based search strategy. Our key algorithmic contribution is showing that an optimal strategy for searching out new passwords from old ones (in our transform-based framework) is NP-hard to develop — one of the few pieces of good news we have to offer defenders — but is also efficiently approximable. We then apply this approximation algorithm to our dataset to generate approximately optimal search strategies, and demonstrate the effectiveness of those search strategies in breaking future passwords from past ones.

The high-order results of our study are alarming, albeit not surprising in light of previous conjectures. We show, for example, that by instantiating our transform-based algorithmic framework with a particular class of transforms, we can break future passwords from past ones in 41% of accounts on average in an offline attack with expected effort of under 3 seconds per account on a 2.67GHz processor. We also show that we can break 17% of accounts on average in an online attack, with fewer than 5 online guesses in expectation. Our study additionally reveals a complex relationship between the susceptibility of accounts to transform-based attacks and the strengths of passwords chosen in those accounts. In other results, our study reveals that the previous use of syntactic transforms in selecting passwords is a strong indicator of their future use: among accounts exhibiting such a previous use of transforms from a class that we will define, we can break future passwords from past ones using the same class of transforms in 63% of accounts on average in an offline attack with a similar level of effort. We also study particular subclasses of transforms; here the results are as much curious as they are alarming. For example, the past substitution of characters by their "leet" equivalents (or vice versa) or by characters residing on the same keyboard keys (e.g., "3" by "#") signals the future use of such substitutions in only 5% of accounts, but predicts the future use of a broader class of substitutions (that we will define) in 75% of accounts.

To summarize, the contributions of our paper are as follows. We provide an algorithmic framework for attacking future passwords from expired ones, show that finding the optimal search order in that framework is NP-hard, and provide an efficient algorithm for generating an approximately optimal search order (§3). We then apply these results to a large, real-world dataset to provide the first analysis of the utility of password expiration for its intended purpose (§4). We close with a discussion of the implications of our study (§5) and then conclude (§6).

2. RELATED WORK

Our study focuses on password choices over time, forced by expiration. Others have focused on the relationships among users' password choices in different scenarios. For example, several studies have examined how users choose passwords for multiple sites during the same time period (e.g., [1, 12, 7, 28]). Since each user is free to choose the same password for many sites, this scenario presumably results in less password variation than the scenario we consider, where the user is precluded from reusing an expired password (in our dataset, for a year; see §4). Shay et al. [22] studied password choices forced by a substantial change in password policy, where one might suspect that users' new passwords would differ more from their old ones than in the scenario we evaluate (where password policy remained constant over time). In addition to exploring a different setting than the above works, our study contributes by providing an algorithmic framework and empirical measurement of the incremental cost of finding new passwords from previous ones. Moreover, unlike studies conducted in a laboratory environment (e.g., [7, 28]) or based on self-reported data (e.g., [1, 22]), ours directly employs user password choices in practice.

There are other hazards of password expiration that we do not consider here. For example, Adams and Sasse [1] and Stanton et al. [25] report that frequent password expiration causes users to write down their passwords or to choose very simple passwords that would be more easily broken by a dictionary attack. Lacking a comparable dataset of passwords for a system that does not perform expiration, we have no baseline against which to evaluate the second claim, in particular. Patterson [21] reported anecdotally that a user circumvented a password expiration system that recorded a fixed number of each user's most recent passwords to prevent their reuse, by changing his password repeatedly until his favorite is cycled off the list and so could be set again. Since the system from which our data was obtained prevents the reuse of a password for a year, it was not vulnerable to such practices.

More distantly related to our work are password strength or memorability studies without specific attention to expiration (e.g., [18, 10, 9, 14]), proposals to help users memorize passwords (e.g., [16, 15, 13]), and proactive checking to force users to choose strong passwords (e.g., [14, 23, 4, 27]). Algorithms for password cracking (absent previous passwords for the same account) has also been an active field of research (e.g., [20, 19, 26]); as we will describe, we utilized some of these techniques in order to initially crack passwords as a precursor to our study (see §4). To our knowledge, however, our study here is the most extensive algorithmic and quantitative analysis to date of attacking new passwords from expired ones.

3. TRANSFORM-BASED ALGORITHMS

As discussed in §2, reports such as that by Adams and Sasse [1] suggest that users often respond to password expiration by transforming their previous passwords in small ways. In this section, we use this insight to develop an algorithmic framework that takes as input an old password σ_k for account k, and that searches for the new password π_k for that account. Our algorithmic framework tries to guess π_k by building from σ_k using a set T of primitive *transforms*. If $\mathcal{P}$ denotes the password space, then each transform $t : \mathcal{P} \to \mathcal{P} \cup \{\perp\}$ is a deterministic algorithm that takes as input a password and that produces a new password or $\perp$ (failure). Intuitively, we think of each transform as making a small modification to an existing password (e.g., change the first "a" to "A"). If the transform is not applicable to the existing password (e.g., the password has no "a"), then the transform produces $\perp$. Let $\mathcal{T} = \bigcup_{\ell=1}^{d} T^\ell$ be the set of all sequences of transforms up to length d, which can be organized as a tree rooted at an additional, empty sequence $\langle\cdot\rangle$ and in which ancestors of any node $\vec{t} \in \mathcal{T}$ are exactly the prefixes of $\vec{t}$. An example such tree is shown in Figure 1.

When searching $\mathcal{T}$ to generate π_k, the adversary visits the nodes of $\mathcal{T}$ in some order $\vec{t}_1\vec{t}_2...$ Visiting a new node $\vec{t}_i$ requires the application of a single additional primitive transform $t \in T$ to extend some $\vec{t}_{i'}$ earlier in the order, i.e., such that $i' < i$. In doing so, the adversary produces a new guess $\vec{t}_i(\sigma_k)$ for π_k. However, because it is possible that $\vec{t}_i(\sigma_k) = \perp$ (i.e., $\vec{t}_i(\sigma_k)$ fails) or $\vec{t}_i(\sigma_k) \in \bigcup_{i'<i}\{\vec{t}_{i'}(\sigma_k)\}$ (i.e., $\vec{t}_i(\sigma_k)$ resulted in a duplicate guess), searching $\mathcal{T}$ generally yields fewer than $|\mathcal{T}|$ unique guesses.

The order in which the adversary searches $\mathcal{T}$ can make a large difference in the performance of the search to find π_k, particularly since the size of $\mathcal{T}$ grows exponentially in d (specifically, $|\mathcal{T}| = ((|T|^{d+1} - 1)/(|T| - 1)) - 1$). In the rest of this section, we explore algorithms for optimizing this order using old passwords $\sigma_{1..n}$ and corresponding new passwords $\pi_{1..n}$ for a collection of accounts $1..n$ as "training data".

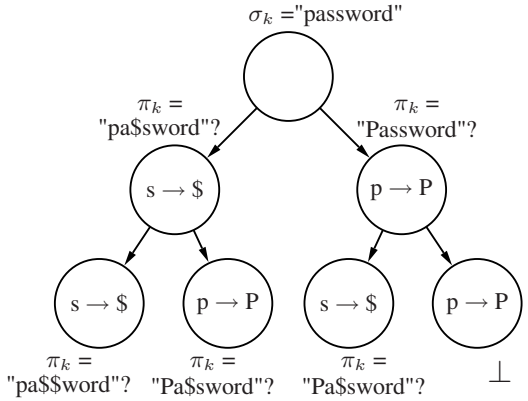


Figure 1: An example transform tree $\mathcal{T}$. Each node represents the transform sequence $\vec{t}$ encountered on the path from the root to this node, which in this case is composed of location-independent transforms (i.e., T_{LI}, see §3.3). Depth is $d = 2$. A search begins at the root with an input password σ_k. Upon visiting a node, the last transform in the corresponding sequence is applied to the output (if not $\perp$) of its parent node. Each output is tested for equality with the target password π_k by hashing it.

More specifically, consider a random account $r \overset{R}{\leftarrow} \{1..n\}$. Let $\vec{t}(\sigma_r) = \pi_r$ denote the event that the sequence $\vec{t} \in \mathcal{T}$, applied in order to a password σ_r, will produce π_r, and so $\mathbb{P}\left(\vec{t}(\sigma_r) = \pi_r\right)$ is the probability of this event under random choice of r. Let $\pi_r \in \mathcal{T}(\sigma_r)$ denote the event $\bigvee_{\vec{t}\in\mathcal{T}} \vec{t}(\sigma_r) = \pi_r$, i.e., that there is some $\vec{t} \in \mathcal{T}$ such that $\vec{t}(\sigma_r) = \pi_r$. The sense in which we seek to optimize the search order for the nodes of $\mathcal{T}$ is to minimize the *expected* number of nodes of $\mathcal{T}$ that need to be searched, under random choice r of account, conditioned on the event $\pi_r \in \mathcal{T}(\sigma_r)$. As such, we consider the following problem:

> EXPECTED MIN TRANSFORM SEARCH (emts): Given is a set T of transforms, a depth d, and collections $\sigma_{1..n}$ and $\pi_{1..n}$ of old and new passwords, respectively, for accounts $1..n$. Let $\mathcal{T} = \bigcup_{\ell=1}^{d} T^{\ell}$, and let $\mathsf{order}_{\mathsf{emts}} : \mathcal{T} \to \{1..|\mathcal{T}|\}$ be a bijection, such that for any distinct $\vec{t}, \vec{u} \in \mathcal{T}$, if $\vec{t}$ is a prefix of $\vec{u}$, then $\mathsf{order}_{\mathsf{emts}}(\vec{t}) < \mathsf{order}_{\mathsf{emts}}(\vec{u})$. The objective is to find $\mathsf{order}_{\mathsf{emts}}$ so as to minimize
>
> $$\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r)) \qquad (1)$$
>
> where $\vec{t}_i = \mathsf{order}^{-1}_{\mathsf{emts}}(i)$ and where the expectation is taken with respect to random selection of $r \overset{R}{\leftarrow} \{1..n\}$.

In words, the EXPECTED MIN TRANSFORM SEARCH problem is to find a search order for $\mathcal{T}$ that minimizes the expected cost of finding π_r from σ_r (when π_r can be generated σ_r, i.e., when $\pi_r \in \mathcal{T}(\sigma_r)$) for a randomly selected account r.

It will be convenient later to have the additional notation

$$\mathsf{cover}_{\mathsf{emts}}(k) = \min_{\vec{t}:\vec{t}(\sigma_k)=\pi_k} \mathsf{order}_{\mathsf{emts}}(\vec{t})$$

for any $\pi_k \in \mathcal{T}(\sigma_k)$. Then, we can equivalently write (1) as

$$\mathbb{E}(\mathsf{cover}_{\mathsf{emts}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))$$

3.1 NP-Hardness of emts

A challenging aspect of EXPECTED MIN TRANSFORM SEARCH is that multiple transform sequences $\vec{t}$ can transform the same input into the same output. One example is shown in Figure 1, where two paths from the root (transform sequences) both produce "`Pa$sword`" from "`password`"; in this case, this occurs because the transform sequences $\vec{t}$ and $\vec{u}$ that produce them are equivalent. This can happen even when $\vec{t}$ and $\vec{u}$ are not equivalent, such as if $\vec{t}$ replaces all instances of "`s`" with "`$`", $\vec{u}$ replaces the first character (whatever it is) with "`$`", and the input password is $\sigma_k =$ `steve79#`.

Such overlap in coverage is characteristic of set cover problems, and in fact we can show the NP-hardness of emts by reducing from the following NP-hard problem:

> MIN SUM SET COVER (mssc) [11]: Given is a set U and a collection $\mathcal{S}$ of subsets of U where $\bigcup_{S\in\mathcal{S}} S = U$. Let $\mathsf{order}_{\mathsf{mssc}} : \mathcal{S} \to \{1..|\mathcal{S}|\}$ be a bijection, and let $\mathsf{cover}_{\mathsf{mssc}} : U \to \{1..|\mathcal{S}|\}$ be defined by
>
> $$\mathsf{cover}_{\mathsf{mssc}}(j) = \min_{S\ni j} \mathsf{order}_{\mathsf{mssc}}(S),$$
>
> i.e., $\mathsf{cover}_{\mathsf{mssc}}(j)$ is the first subset in the ordering to contain j. The objective is to find $\mathsf{order}_{\mathsf{mssc}}$ so as to minimize $\sum_{j\in U} \mathsf{cover}_{\mathsf{mssc}}(j)$.

Given an instance $(U, \mathcal{S})$ of MIN SUM SET COVER, denote $U = \{1..n\}$. We reduce this instance of mssc to an equivalent instance of emts by creating, for each $j \in U$, an "account" with a pair of old and new passwords, and for each $S \in \mathcal{S}$, a transform that maps the old passwords for the accounts corresponding to its elements to their new passwords. Specifically, for each j, create an old password σ_j and a new password π_j, such that $\sigma_{1..n}$ and $\pi_{1..n}$ comprise $2n$ distinct passwords. For each $S \in \mathcal{S}$, create a primitive transform t_S such that $t_S(\sigma_j) = \pi_j$ if $j \in S$ and such that t_S fails on any other input. The set of $|\mathcal{S}|$ such primitive transforms comprise the set T. Set depth $d = 1$.

Consider any $\mathsf{order}_{\mathsf{emts}}$ for searching $\mathcal{T}$. Set $\mathsf{order}_{\mathsf{mssc}}(S) \leftarrow \mathsf{order}_{\mathsf{emts}}(\langle t_S \rangle)$. Then,

$$\begin{aligned}
&\mathbb{E}(\mathsf{cover}_{\mathsf{emts}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r)) \\
&= \mathbb{E}\left(\mathsf{cover}_{\mathsf{emts}}(r)\right) \text{ since } \pi_k \in \mathcal{T}(\sigma_k) \text{ for all } k \\
&= \sum_{i=1}^{n} i \cdot \mathbb{P}\left(\mathsf{cover}_{\mathsf{emts}}(r) = i\right) \\
&= \sum_{i=1}^{n} i \cdot \frac{|\{k \in \{1..n\} : \mathsf{cover}_{\mathsf{emts}}(k) = i\}|}{n} \\
&= \sum_{i=1}^{n} i \cdot \frac{|\{j \in U : \mathsf{cover}_{\mathsf{mssc}}(j) = i\}|}{n} \\
&= \frac{1}{n} \sum_{j\in U} \mathsf{cover}_{\mathsf{mssc}}(j)
\end{aligned}$$

Thus, $\mathsf{order}_{\mathsf{emts}}$ minimizes $\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ if and only if $\mathsf{order}_{\mathsf{mssc}}$ minimizes $\sum_{j\in U} \mathsf{cover}_{\mathsf{mssc}}(j)$.

3.2 Approximation Algorithm for emts

Feige et al. [11] provided an efficient greedy algorithm $\mathcal{B}_{\mathsf{mssc}}$ that is a 4-approximation for mssc. Specifically, $\mathcal{B}_{\mathsf{mssc}}$ defines its order $\mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}$ as follows: $\mathsf{order}^{-1}_{\mathcal{B}_{\mathsf{mssc}}}(i)$ is the set S that includes the most elements of U that are not included in $\bigcup_{i'<i} \mathsf{order}^{-1}_{\mathcal{B}_{\mathsf{mssc}}}(i')$.

The algorithm $\mathcal{B}_{\mathsf{mssc}}$ can be used to build a $4d$-approximation algorithm $\mathcal{B}_{\mathsf{emts}}$ for emts, as follows. Define for each $\vec{t} \in \mathcal{T}$ the set $S_{\vec{t}} \leftarrow \{k : \vec{t}(\sigma_k) = \pi_k\}$, and let $\mathcal{S} \leftarrow \{S_{\vec{t}}\}_{\vec{t}\in\mathcal{T}}$ and $U \leftarrow \bigcup_{\vec{t}\in\mathcal{T}} S_{\vec{t}}$. $\mathcal{B}_{\mathsf{mssc}}(U, \mathcal{S})$ then induces an order $\mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}$ on these sets and, in turn, the corresponding transform sequences; if $i = \mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}(S_{\vec{t}})$, then denote $\vec{t}$ by $\vec{u}_i$. Note that $\vec{u}_1, \vec{u}_2, ..$, however, might not constitute a feasible search order for $\mathcal{T}$, since each $\vec{u}_i$

might not be preceded by its prefixes. The algorithm $\mathcal{B}_{\mathsf{emts}}$ thus works by inserting the prefixes of $\vec{u}_i$ just before $\vec{u}_i$, as needed.

Algorithm $\mathcal{B}_{\mathsf{emts}}(T, d, \sigma_{1..n}, \pi_{1..n})$:
1: $\mathcal{T} \leftarrow \bigcup_{\ell=1}^{d} T^{\ell}$
2: **for** $\vec{t} \in \mathcal{T}$ **do**
3: $\quad S_{\vec{t}} \leftarrow \{k : \vec{t}(\sigma_k) = \pi_k\}$
4: $U \leftarrow \bigcup_{\vec{t} \in \mathcal{T}} S_{\vec{t}}$; $\mathcal{S} \leftarrow \{S_{\vec{t}}\}_{\vec{t} \in \mathcal{T}}$
5: $\mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}} \leftarrow \mathcal{B}_{\mathsf{mssc}}(U, \mathcal{S})$
6: $i \leftarrow 0$
7: **for** $i' = 1..|\mathcal{T}|$ **do**
8: $\quad \vec{t} \leftarrow \vec{u} : \mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}(S_{\vec{u}}) = i'$
9: $\quad$ **for** $i'' = 1..|\vec{t}|$ **do**
10: $\quad\quad$ **if** $\vec{t}[1..i''] \notin \{\mathsf{order}^{-1}_{\mathcal{B}_{\mathsf{emts}}}(1)..\mathsf{order}^{-1}_{\mathcal{B}_{\mathsf{emts}}}(i)\}$ **then**
11: $\quad\quad\quad i \leftarrow i + 1$
12: $\quad\quad\quad \mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}(\vec{t}[1..i'']) \leftarrow i$
13: **return** $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$

Figure 2: Search algorithm $\mathcal{B}_{\mathsf{emts}}$

More specifically, $\mathcal{B}_{\mathsf{emts}}$ creates a new order $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$ as shown in Figure 2. It first queries $\mathcal{B}_{\mathsf{mssc}}(U, \mathcal{S})$ (line 5) using U and $\mathcal{S}$ created as described above (lines 2–4). It then steps through the nodes of $\mathcal{T}$ in the order that $\mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}$ prescribes for their corresponding sets $\mathcal{S}$ (lines 7–8). For each $\vec{t}$ considered, the algorithm inserts any missing prefixes of $\vec{t}$ (lines 9–12) and, finally, $\vec{t}$ itself (lines 9–12 when $i'' = |\vec{t}|$). Note that in line 10, the notation $\vec{t}[1..i'']$ denotes the length-i'' prefix of $\vec{t}$.

For any k such that $\pi_k \in \mathcal{T}(\sigma_k)$, define

$$\mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k) = \min_{S_{\vec{t}} \ni k} \mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}(S_{\vec{t}})$$

$$\mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(k) = \min_{\vec{t} : \vec{t}(\sigma_k) = \pi_k} \mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}(\vec{t})$$

and let $\mathsf{cover}^*_{\mathsf{mssc}}$ and $\mathsf{cover}^*_{\mathsf{emts}}$ denote the functions $\mathsf{cover}_{\mathsf{mssc}}$ and $\mathsf{cover}_{\mathsf{emts}}$ resulting from optimal solutions to mssc instance $(U, \mathcal{S})$ and emts instance $(T, d, \sigma_{1..n}, \pi_{1..n})$, respectively. Then,

$$\begin{aligned}
&\frac{\mathbb{E}(\mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))}{\mathbb{E}(\mathsf{cover}^*_{\mathsf{emts}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))} \\
&= \frac{\mathbb{E}(\mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))}{\mathbb{E}(\mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))} \cdot \frac{\mathbb{E}(\mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))}{\mathbb{E}(\mathsf{cover}^*_{\mathsf{emts}}(r) \mid \pi_r \in \mathcal{T}(\sigma_r))} \\
&= \frac{\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(k)}{\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k)} \cdot \frac{\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k)}{\sum_k \mathsf{cover}^*_{\mathsf{emts}}(k)} \\
&\le \frac{\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(k)}{\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k)} \cdot \frac{\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k)}{\sum_k \mathsf{cover}^*_{\mathsf{mssc}}(k)} \qquad (2) \\
&\le d \cdot 4 \qquad (3)
\end{aligned}$$

where the sums are taken over all k such that $\pi_k \in \mathcal{T}(\sigma_k)$. Above, (2) follows because $\sum_k \mathsf{cover}^*_{\mathsf{mssc}}(k) \le \sum_k \mathsf{cover}^*_{\mathsf{emts}}(k)$. (3) holds since $\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k) \le 4 \cdot \sum_k \mathsf{cover}^*_{\mathsf{mssc}}(k)$ [11], and because for any $\vec{t}$, $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}(\vec{t}) \le d \cdot \mathsf{order}_{\mathcal{B}_{\mathsf{mssc}}}(S_{\vec{t}})$, since $\mathcal{B}_{\mathsf{emts}}$ may insert up to d nodes of $\mathcal{T}$ before each node $\vec{t}_i$ output by $\mathcal{B}_{\mathsf{mssc}}$. ($|\vec{t}|$ in line 9 is at most d.) Therefore, $\sum_k \mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(k) \le \sum_k d \cdot \mathsf{cover}_{\mathcal{B}_{\mathsf{mssc}}}(k)$. So, $\mathcal{B}_{\mathsf{emts}}$ is a $4d$-approximation for emts.

The time complexity of $\mathcal{B}_{\mathsf{mssc}}(U, \mathcal{S})$ is $O(|U| \cdot |\mathcal{S}|)$. As used in $\mathcal{B}_{\mathsf{emts}}(T, d, \sigma_{1..n}, \pi_{1..n})$, where $|U| \le n$ and $|\mathcal{S}| = |\mathcal{T}|$, its complexity is thus $O(n|\mathcal{T}|)$. $\mathcal{B}_{\mathsf{emts}}$ also performs up to d loop iterations per $\vec{t} \in \mathcal{T}$, effectively walking $\mathcal{T}$ from its root to $\vec{t}$ (lines 9–12). Consequently, the time complexity of $\mathcal{B}_{\mathsf{emts}}(T, d, \sigma_{1..n}, \pi_{1..n})$ is $O(n|\mathcal{T}| + d|\mathcal{T}|)$. Finally, because $|\mathcal{T}| = (|T|^{d+1} - 1)/(|T| - 1) - 1 = O(|T|^d)$, the complexity of this algorithm is $O((n + d)|T|^d)$.

T	$\lvert T\rvert$	$\lvert\mathcal{T}\rvert$			
		$d=1$	$d=2$	$d=3$	$d=4$
T_{ED}	3402	3402	11577006	3.9×10^{10}	1.3×10^{14}
T_{EDM}	4371	4371	19110012	8.4×10^{10}	3.7×10^{15}
T_{LI}	534	534	285690	152558994	8.1×10^{10}
T_{LIP}	50	50	2550	127550	6377550

Figure 3: Sizes of transform sets and resulting trees

3.3 Instantiating $\mathcal{B}_{\mathsf{emts}}$ with Transforms

We consider the following sets T of transforms. Figure 3 shows the sizes of these sets and the trees that result at different depths d.

- **Edit distance**: The *edit distance* between two strings is the minimum number of character insertions, deletions or replacements necessary to turn one string into the other. For our analysis, the transforms for $T = T_{\mathsf{ED}}$ that we apply to an input σ of length ℓ include character deletion, insertion, and replacement at a specific position. The number of position-dependent transforms in T_{ED} thus depends on ℓ. In our evaluations, we constructed T_{ED} to accommodate password lengths up to $\ell = 18$, as this accommodated all password lengths that occurred in our data (see §4).
- **Edit distance with substring moves**: Edit distance with substring moves [8] is a variation of edit distance that permits a substring move in one step. The transforms $T = T_{\mathsf{EDM}}$ in this case include all of T_{ED} in addition to:
  - A substring move with parameters $1 \le j \le j' \le j'' \le \ell$ results in $\sigma[1..(j-1)]\sigma[j'..j'']\sigma[j..(j'-1)]\sigma[(j''+1)..\ell]$.

  For example, `password` could be changed to `wordpass` in a single substring move (with $j = 1$, $j' = 5$, and $j'' = 8$).
- **Hand-crafted location-independent transforms**: We also consider a set $T = T_{\mathsf{LI}}$ that, unlike the case of edit distance with or without moves, can be applied at any location in a password. The types of such transforms that we include in T_{LI} cover eight disjoint categories:
  - T_{cap}: capitalization (e.g., "`17candy#`" → "`17candY#`")
  - T_{del}: digit and special character deletion (e.g., "`alex28!!!`" → "`alex28!!`")
  - T_{dup}: digit and special character duplication (e.g., "`stinson1!`" → "`stinson11!`")
  - T_{sub}: digit and special character substitution with the same character type (e.g., "`tar!heel1`" → "`tar!heel2`")
  - T_{ins}: sequential insertion (e.g., "`dance#7`" → "`dance#78`")
  - T_{leet}: *leet* transformation (e.g., "`raven#1&`" → "`r@ven#1&`")
  - T_{mov}: letter, digit or special character block moves (e.g., "`$steve27`" → "`27$steve`")
  - T_{key}: replacement of a digit or special character with the alternate character for the same key (e.g., "`100py*!2`" → "`100py*!@`")

  In total, we derived 534 location-independent transforms in T_{LI}. For completeness, the full list is provided in Appendix A. Given time constraints, it was not possible to apply these transforms beyond $d = 3$. However, to explore the impact of expanding our search to larger values of d, we consider one final category.
- **Pruned hand-crafted location-independent transforms**: We selected the 50 most successful transforms $T_{\mathsf{LIP}} \subseteq T_{\mathsf{LI}}$ at $d = 1$. The specifics of how we choose this subset is discussed in Appendix A. Given this reduced set T_{LIP}, we were able to search to $d = 4$ in our experiments.

4. EVALUATION

For this study, we examine password hashes for accounts of the ONYEN (`http://onyen.unc.edu`) single-sign-on system at our institution. Each member of the university community is assigned an ONYEN (an acronym for "Only Name You'll Ever Need"). The password for each ONYEN is required to change every 3 months; ONYENs for which this change does not occur are suspended. The password management policy requires a user to follow the following rules when creating a new password for an ONYEN:

- It cannot have been used for this ONYEN in the last year.
- It must be at least 8 characters long.
- It must contain at least one letter and at least one digit.
- It must contain at least one of the following special characters: `!@#$%&*+={}?<>"'`
- It must share fewer than six (or length of the ONYEN, if less than six) consecutive common characters with the ONYEN.
- It must not start with a hyphen, end with a backslash, start or end with a space, or contain a double-quote anywhere except as the last character.

The dataset we acquired contains 51141 unsalted MD5 password hashes from 10374 defunct ONYENs (used between 2004 and 2009), with 4 to 15 password hashes per ONYEN, i.e., the hashes of the passwords chosen for that ONYEN sequentially in time. The ONYENs themselves were not provided with the passwords, and so we have no knowledge of the users to whom these passwords corresponded. However, since ONYENs are broadly used by UNC faculty, staff, and students, and employees of UNC hospitals, we believe that this data reflects a diversity of user educations and backgrounds. The data collected represents a time span during which the password management policy was the same as it is today. Another pertinent fact is that ONYENs are widely used at UNC for private services such as email, access to payroll management, benefits selection, etc. As such, ONYENs play a significant role in users' daily lives, in contrast to seldomly used web-based accounts. Moreover, because the ONYEN is required for gaining access to sensitive information (e.g., payroll) users have strong incentives for choosing "good" passwords.

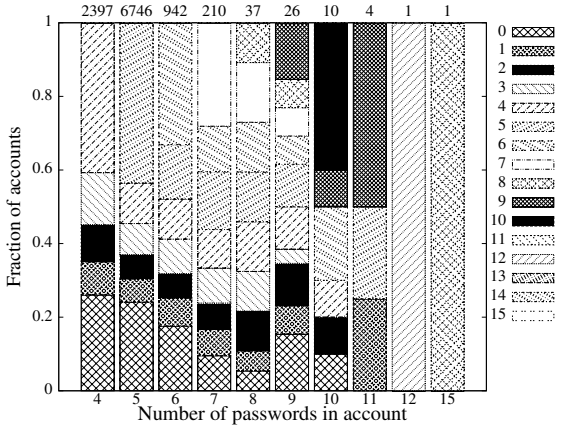


Figure 4: Passwords cracked per account in our dataset. Each bar represents accounts with the number of passwords indicated on the x-axis. The number of such accounts is shown above each bar. Regions within each bar show the fraction of these accounts for which the indicated number of passwords were cracked.

Since we were provided only hashes, the first challenge was to crack as many of these passwords as we could. Due to confidentiality agreements that accompanied this data, we were unable to distribute this data widely to many machines. Instead, all of our cracking activity that involved accessing these hashes was isolated on two specially configured machines, each with two quad-core 2.67GHz processors and 72GB of RAM. One computer was available for this activity starting mid-October 2009, and the second came online in January 2010, though neither could be exclusively dedicated to password cracking.

We employed various approaches to crack passwords: dictionary-based password cracking using "John the Ripper" (`http://www.openwall.com/john/`), including its "Markov mode" provided as a patch for version 1.7.6; brute-force password cracking; and rainbow tables [20]. The dictionary-based approach was most effective, especially when combined with the word-list-generating method of Weir et al. [26]. For passwords we cracked via these techniques, we further attempted to crack other passwords for the same ONYEN using the techniques we described in §3. (We will evaluate the effectiveness of those techniques in §4.1–4.2.)

At the time of this writing, we have cracked 31075 passwords for 7936 ONYENs. Figure 4 shows the fraction of passwords cracked for each ONYEN, with ONYENs separated by the number of hashes available for it. For example, as illustrated in the left-most column, which describes accounts with four passwords, we broke no passwords in 25% of these accounts; one password in 10%; two passwords in 10%; three passwords in 15%; and all four passwords in 40%. Overall, among the 7936 ONYENs in which we cracked at least one password, we broke all passwords belonging to 54% of these ONYENs, and broke at least half in 90%.

Since our goals specifically focus on guessing future passwords from past ones for the same ONYENs, we restrict our attention to only those ONYENs for which we have at least one cracked password and, among ONYENs with only one cracked password, those in which the cracked password is not the last one in the account. In the rest of this paper, we use the $n = 7752$ ONYENs meeting this criterion as our experimental data. For such accounts, though, even passwords we have *not* cracked but that temporally follow a cracked password can be useful in our evaluations. For example, in §4.1 we define each σ_k to be a password that has been cracked and the hash for which is not the last for its ONYEN, and π_k to be the password corresponding to a hash for the same ONYEN that came temporally after that of σ_k (but not necessarily immediately), cracked or not. Then, given σ_k, we can determine whether $\pi_k \in \mathcal{T}(\sigma_k)$, even if we have not cracked π_k. More to the point, if we have not cracked π_k, then this implies that $\pi_k \notin \mathcal{T}(\sigma_k)$. In the case that we have cracked π_k, then we can obviously determine whether $\pi_k \in \mathcal{T}(\sigma_k)$ and, if so, the value of $\mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(k)$.

4.1 Evaluation Over All Accounts

We now evaluate the effectiveness of the $\mathcal{B}_{\mathsf{emts}}$ approach in breaking passwords. To do so, we perform a series of *trials*; in each, $\sigma_{1..n}$ and $\pi_{1..n}$ are fixed. To instantiate $\sigma_{1..n}$ and $\pi_{1..n}$ for a trial, we populate σ_k with a password from account k chosen uniformly at random from those we have cracked, excluding the last password for the account. We then instantiate π_k with a password (cracked or uncracked) from account k chosen uniformly at random from those that followed σ_k temporally. As discussed in §4, if we have not cracked π_k, this implies $\pi_k \notin \mathcal{T}(\sigma_k)$ (for any $\mathcal{T}$ we consider), and so such a password pair contributes to the probability of event $\pi_r \notin \mathcal{T}(\sigma_r)$ under random selection of r. For any such instantiation of $\sigma_{1..n}$ and $\pi_{1..n}$, we then conduct a *trial* as defined below. The numbers we report are the average of at least 10 trials.

In each trial, we partition the indices $1..n$ into five blocks, and then perform a five-fold cross validation; i.e., we perform tests in

T	d	$\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$	**Algorithm $\mathcal{B}_{\mathsf{emts}}$ (§3.2)** $\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ (skipped, failed, viable)	**Breadth-first search** $\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ (skipped, failed, viable)
T_{ED}	1	0.26	145.29 (0.00, 33.41, 111.88)	740.03 (0.00, 64.12, 675.91)
	2	0.39	284790.10 (37054.93, 112244.51, 135490.65)	562986.90 (48461.52, 238323.80, 276202.00)
T_{EDM}	1	0.28	224.51 (0.00, 69.22, 155.29)	913.59 (0.00, 168.79, 744.81)
	2	0.41	481607.44 (101137.59, 206639.18, 173830.67)	851020.60 (126514.20, 399700.70, 324805.60)
T_{LI}	1	0.25	65.52 (0.00, 53.21, 12.31)	261.57 (0.00, 220.15, 41.42)
	2	0.37	15534.08 (13022.51, 2188.12, 323.44)	33293.50 (28034.53, 4627.74, 631.23)
	3	0.41	3082677.88 (3021178.93, 53122.41, 8376.54)	3504117.38 (3432836.84, 61667.63, 9612.92)
T_{LIP}	1	0.17	17.35 (0.00, 13.15, 4.21)	16.63 (0.00, 12.55, 4.08)
	2	0.24	84.49 (45.87, 28.41, 10.20)	326.39 (239.57, 66.91, 19.92)
	3	0.28	2543.04 (2366.45, 131.52, 45.06)	5630.27 (5256.80, 283.92, 89.55)
	4	0.30	91952.11 (90267.93, 1211.53, 472.65)	199697.40 (196940.90, 2035.80, 720.76)

Figure 5: Evaluation of all accounts (§4.1). Each value is an average over 10 trials.

which a different block is used as testing data after training on the other four. More specifically, in each test the four "training" blocks are used to select the order in which the nodes of $\mathcal{T}$ are searched, per algorithm $\mathcal{B}_{\mathsf{emts}}$. Then, each (σ_k, π_k) in the "testing" block is checked to determine if $\pi_k \in \mathcal{T}(\sigma_k)$ and, if so, the value of $\mathsf{cover}_{\mathcal{B}_{\mathsf{emts}}}(k)$. This allows us to compute $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ and $\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ for this trial. We also dissect this expected value, reporting the average number of nodes $\vec{t}$ (prior to finding π_k) that were *skipped* because for some strict prefix $\vec{u}$ of $\vec{t}$, $\vec{u}(\sigma_k) = \perp$; that *failed* because $\vec{t}(\sigma_k) = \perp$; and that were *viable* in that $\vec{t}(\sigma_k) \neq \perp$.

To demonstrate the cost savings in breaking passwords offered by $\mathcal{B}_{\mathsf{emts}}$, we also show in Figure 5 the analogous costs if the tree $\mathcal{T}$ were searched using breadth-first search. Here we see that $\mathcal{B}_{\mathsf{emts}}$ offers significant cost savings, reducing $\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ from that achieved by breadth-first search by roughly 75% or more when $d = 1$ and by roughly 45% or more when $d = 2$. The one exception in the $d = 1$ case is $T = T_{\mathsf{LIP}}$; this is due to the fact that T_{LIP} was already chosen to include only those transforms t yielding the largest $\mathbb{P}(\pi_r = t(\sigma_r))$ (see Appendix A) and are chosen in decreasing order of that value. And, while the advantages of $\mathcal{B}_{\mathsf{emts}}$ diminish for $d = 3$ in the case $T = T_{\mathsf{LI}}$, we conjecture that this is due to a lack of sufficient training data for such a large transform search tree. Note that the improvement offered by $\mathcal{B}_{\mathsf{emts}}$ for $T = T_{\mathsf{LIP}}$ remains above 50% through $d = 4$.

One might argue that even given the higher cost of searching $\mathcal{T}$ using a breadth-first search strategy, doing so might still be practical for the trees $\mathcal{T}$ considered in Figure 5. Additionally, producing an optimized search order via $\mathcal{B}_{\mathsf{emts}}$ is a nontrivial computation that grows as the search trees and training datasets get larger. Breadth-first search imposes no such up-front cost.

While true, for larger trees than we have considered so far, the one-time cost of $\mathcal{B}_{\mathsf{emts}}$ should be more than offset by the cost savings that $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$ offers per account attacked. It also enables one to more effectively short-circuit tree searches early. For example, for the case $T = T_{\mathsf{LI}}$ and $d = 2$, we find that 80% of accounts that will be broken by $\mathcal{T}$ will be broken in the first 620 elements of $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$, or after searching only about 0.2% of the tree. To crack the same fraction of accounts, breadth-first search explores about 110000 elements (about 40% of the tree). The cost savings of $\mathcal{B}_{\mathsf{emts}}$ are particularly important for online attacks, where password guesses are limited; we will discuss these below.

Implications for offline attacks.

There are two contexts in which it makes sense to interpret the results shown in Figure 5, corresponding to the two contexts in which passwords are typically used. We first consider passwords that can be subjected to an offline attack; for example, a password may be used to encrypt files, and the attacker would like to retain access to files encrypted under a password following a password change. In this case, the adversary, knowing σ_k and having access to files encrypted under π_k, faces an offline attack to find π_k. The actual runtimes, on average, to break π_k in such a situation with the trees $\mathcal{T}$ considered in Figure 5 are shown in Figure 6. Skipped, failed and viable nodes in $\mathcal{T}$ do not contribute equally to these runtimes: skipped nodes cost only the time to discard or avoid them; failed nodes $\vec{t}$ cost the time to attempt the last transform in the sequence $\vec{t}$ (on a previously computed result); and viable nodes $\vec{t}$ cost the time to apply the last transform and to hash the resulting password, to test against that of π_k.

Arguably the key takeaway from this figure, however, is that even the most expensive password cracking effort ($T = T_{\mathsf{LI}}$, $d = 3$) required an average of only under 3 seconds per password that it broke. In combination with the success rate $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ for this configuration (see Figure 5) we reach a fairly alarming conclusion: *On average, roughly 41% of passwords π_k can be broken from an old password σ_k in under 3 seconds.*

Implications for online attacks.

The second context in which to consider the results of Figure 5 is an online attack, in which the attacker knows σ_k and must submit online guesses to a server in order to break π_k. Many servers are configured to permit only few wrong password guesses before "locking" the account, and so the relevant measure becomes the fraction of accounts the attacker can break in the initial several viable guesses when examining nodes in order of $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$. Figure 5 indicates, for example, that using T_{LIP} with $d = 1$, *an average of* 17% *of accounts can be broken in under five online password guesses in expectation*. Figure 7 provides a more refined view into the effectiveness of each transform search for an online attack; each graph shows the average fraction of passwords cracked for a given number of viable guesses produced by searching the specified transform set T to the indicated depth d. For example, Figure 7(d) shows that an average of 13% of accounts can be broken (with certainty) in 5 online guesses, and 18% can be broken in 10 guesses.

Password strength versus susceptibility to transform-based search.

It is tempting to assume that the new passwords that are most susceptible to transform-based search from old passwords are those that are intrinsically weak. After all, the same "laziness" that causes a user to simply replace a character or two in σ_k to create π_k would,

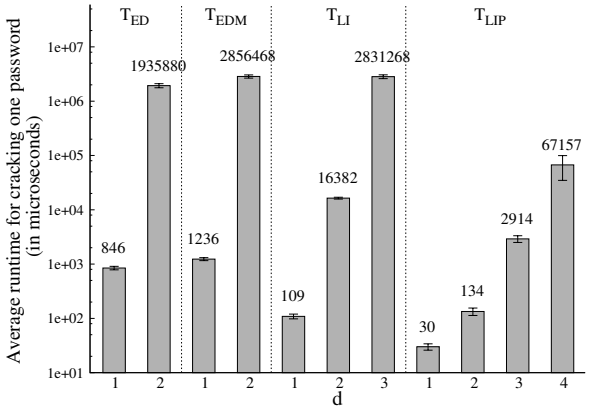
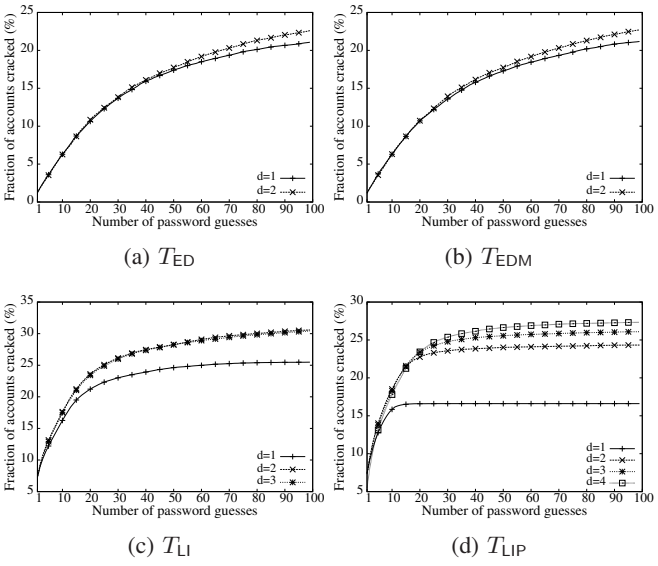


Figure 6: Average runtime to crack a password using $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$ (microseconds on a 2.67GHz processor). Average over 10 trials, with one standard deviation shown.

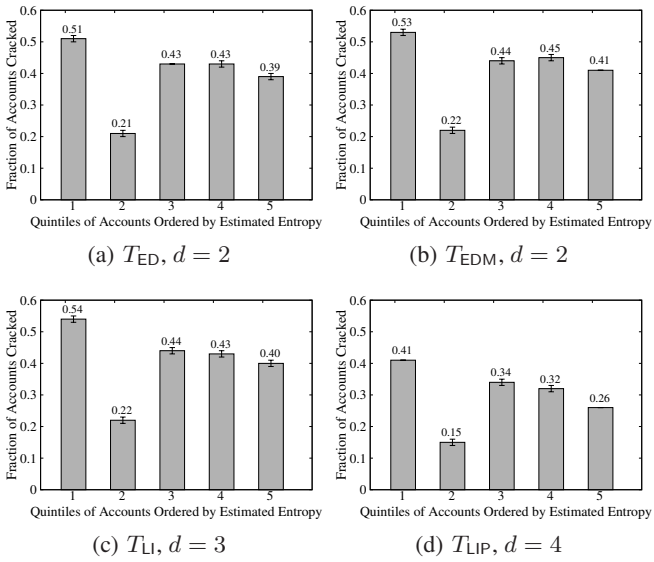


(a) T_{ED} (b) T_{EDM}

(c) T_{LI} (d) T_{LIP}

Figure 7: Fraction of passwords π_k found as function of viable guesses $\vec{t}(\sigma_k)$ made in order of $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$. Average of 10 trials.

it might seem, cause the user to select passwords that are generally weaker when viewed in isolation. To test this conjecture, we categorized the 7752 accounts in our data according to the average estimated entropy of the passwords for that account that we were able to crack. To perform this calculation, we estimated the entropy of each password using the NIST methodology [6]. Among other rules, this methodology prescribes adding six bits to the estimated entropy of a password if the password survives a dictionary attack. Due to the inclusion of nonalphabetic characters in our passwords, none of these passwords would be present in a dictionary of words consisting of only alphabetic characters, and so we awarded these six bits of entropy to a password only if it survived the dictionary attack after removing its nonalphabetic characters. The dictionary we used was derived from the en_US dictionary in Hunspell (Kevin's Word List Page: `http://wordlist.sourceforge.net/`). After converting all uppercase characters in the original dictionary to lowercase (all alphabetic password characters were converted to lowercase, as well) and deleting all purely numeric entries, the dictionary had 49875 words. Testing with other dictionaries did not significantly alter our results.



(a) T_{ED}, $d = 2$ (b) T_{EDM}, $d = 2$

(c) T_{LI}, $d = 3$ (d) T_{LIP}, $d = 4$

Figure 8: $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ per quintile of accounts ordered by entropy (average estimated entropy of passwords in account). The average account entropies per quintile are 19.21, 22.07, 24.01, 25.32 and 28.02. Average of 10 trials, with one standard deviation shown.

In Figure 8 we show the susceptibility of accounts to transform-based search as a function of password strength in the accounts. Each figure represents results averaged over 10 trials, in which an old password σ_k and new password π_k were chosen for each account k in the same way as in our previous experiments. To produce these figures, the accounts were ordered in increasing order according to the average estimated entropy of the passwords in the account and then divided into quintiles. Each bar in Figure 8 corresponds to one quintile of accounts, and shows the fraction of those accounts that succumbed to transform-based search (i.e., $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$, with r chosen at random from that quintile).

These graphs suggest that the weakest passwords are, in fact, the most susceptible to transform-based search, in that the first quintile has the largest fraction of accounts broken. This lends credibility to the intuition that laziness in initial password selection is correlated with laziness in selecting a new password after expiration. To our surprise, however, the fraction of accounts susceptible to transform-based search does not monotonically decrease as a function of average estimated entropy per account, but rather reaches its minimum in the second quintile. One possible explanation for the increase in the third through fifth quintiles is that the increased mental effort required to choose a good password discourages these users from investing that effort again to generate a completely new password after expiration. If true, then increasing the strength requirements on individual passwords may diminish the utility of expiration further. Additional tests are required to validate this, however.

4.2 Accounts with History of Transform Use

We now focus our attention on accounts that previously exhibited selection of a new password by applying transforms to a previous one, to evaluate the extent to which past use of transforms predicts future use. If this extent is substantial, then an adversary with knowledge of past use of transforms within an account may

T	d	Accounts filtered using $\mathcal{T}_{\mathsf{past}} = \mathcal{T}$			n randomly selected accounts	
		n	$\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$	$\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ (skipped, failed, viable)	$\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$	$\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ (skipped, failed, viable)
T_{ED}	1	3412	0.52	130.61 (0.00, 28.02, 102.59)	0.61	193.62 (0.00, 41.37, 152.25)
	2	4530	0.61	284499.41 (33964.39, 113450.82, 137084.20)	0.61	331665.28 (41485.51, 132077.65, 158102.12)
T_{EDM}	1	3729	0.50	203.78 (0.00, 62.37, 141.40)	0.50	285.10 (0.00, 92.51, 192.59)
	2	4679	0.61	470390.25 (88885.91, 207050.44, 174453.89)	0.61	575157.53 (121778.91, 245913.91, 207464.71)
T_{LI}	1	3417	0.50	67.08 (0.00, 55.12, 11.96)	0.50	76.58 (0.00, 62.67, 13.91)
	2	4292	0.60	16395.12 (13743.61, 2312.55, 338.96)	0.60	18797.81 (15779.08, 2633.67, 385.06)
	3	4532	0.63	3321484.36 (3253944.90, 58255.08, 9284.39)	0.63	3353629.77 (3285529.91, 58768.99, 9330.87)
T_{LIP}	1	2678	0.39	17.68 (0.00, 13.71, 3.97)	0.39	18.20 (0.00, 13.78, 4.41)
	2	3406	0.48	84.38 (45.59, 29.02, 9.76)	0.48	113.66 (69.22, 32.88, 11.56)
	3	3608	0.52	2661.70 (2472.10, 141.81, 47.80)	0.52	3243.70 (3025.68, 163.51, 54.51)
	4	3721	0.55	96135.19 (94500.19, 1187.15, 447.86)	0.55	114179.84 (112205.57, 1427.06, 547.22)

Figure 9: Evaluation of accounts with history of transform use (§4.2) using algorithm $\mathcal{B}_{\mathsf{emts}}$. Averages over 10 trials.

focus his attention on retaining access to this account (in lieu of others) across password changes, owing to the increased likelihood of transform use again. We are also interested in learning the extent to which past uses of certain transforms predicts future use of others. For example, if a user previously transformed an old password σ'_k to a subsequent password π'_k by replacing a "o" with "0", then perhaps this user substituted an "i" with "1" when generating his current password π_k from from his previous one σ_k.

Our framework for evaluating the additional utility to the attacker of past transform use works as follows. Consider a set T_{past} of primitive transforms, which yield a depth-d transform tree $\mathcal{T}_{\mathsf{past}}$. We restrict our attention to accounts k such that there exist cracked passwords σ'_k and π'_k such that $\pi'_k \in \mathcal{T}_{\mathsf{past}}(\sigma'_k)$; all other accounts are filtered out. Let the remaining accounts be renumbered $1..n$, where n now denotes the number of remaining accounts. We then repeat the analysis of §4.1 identically using a set T of primitive transforms, but using this filtered set of data, and with the additional caveat that when selecting $\sigma_{1..n}$ and $\pi_{1..n}$ for a trial, π_k must occur temporally after some σ'_k and π'_k satisfying $\pi'_k \in \mathcal{T}_{\mathsf{past}}(\sigma'_k)$. Because we operate on a filtered set of accounts depending on T_{past}, n is potentially different in each case.

We begin by performing this analysis for $\mathcal{T}_{\mathsf{past}} = \mathcal{T}$, for trees $\mathcal{T}$ defined by the same transform sets T considered in §4.1 (i.e., $T \in \{T_{\mathsf{ED}}, T_{\mathsf{EDM}}, T_{\mathsf{LI}}, T_{\mathsf{LIP}}\}$) and the same depths d considered there. In this way, we measure the utility of the preceding use of transforms in T in predicting their future use. The results of this study are shown in Figure 9. It is evident from that these filtered accounts have a significantly higher probability of being broken by searching using $\mathcal{T}$, as can be seen by comparing the columns labeled $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ in Figures 5 and 9. Put another way, if an account contains passwords in which one (π'_r) is created from another (σ'_r) by applying some $\vec{t} \in \mathcal{T}_{\mathsf{past}}$, then future passwords for this account (π_r) are more likely to be created by applying transforms again to some old password σ_r (i.e., $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ is higher for $\mathcal{T} = \mathcal{T}_{\mathsf{past}}$). In some cases the effect is so strong that well over *half* of the accounts exhibit it; e.g., the case in which $T = T_{\mathsf{LI}}$ and $d = 3$ yields $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r)) = .63$.

Somewhat surprisingly, though, $\mathbb{E}(\min\{i : \vec{t}_i(\sigma_r) = \pi_r\} \mid \pi_r \in \mathcal{T}(\sigma_r))$ does *not* show a consistent improvement in Figure 9 over that in Figure 5. The reason is that filtering the accounts using $\mathcal{T}_{\mathsf{past}}$ reduces the number n of accounts to roughly half of the accounts used in the analysis of Figure 5. The corresponding reduction in the training data during the 5-fold cross validation tests causes a decay in the quality of $\mathsf{order}_{\mathcal{B}_{\mathsf{emts}}}$ output by $\mathcal{B}_{\mathsf{emts}}$. To demonstrate this effect, we repeated our tests on n accounts selected uniformly at random from the set used in the Figure 5 tests, and show the results for such tests in Figure 9 under the heading "n randomly selected accounts". We now see that the n accounts chosen by filtering using $\mathcal{T}_{\mathsf{past}}$ are, in fact, less costly to attack than random samples of the same number n of accounts from the data used in Figure 5.

The consequences of this analysis for offline and online attacks are noteworthy. For an offline attack, where an attacker possesses σ_k and can check guesses at π_k unmitigated, it can break π_k using a transform-based search typically in a majority (e.g., 63% in the case of $T = T_{\mathsf{LI}}, d = 3$) of accounts in which the user previously used the same transforms to generate a new password π'_k from a past password σ'_k. Moreover, the speed with which the attacker can do so is comparable to, or faster than, that indicated in Figure 6. For an online attack, where the attacker must submit guesses at π_k to a server, the success rates for the attacker on these accounts is also enhanced, e.g., reaching 39% attack success in the first 4 viable password guesses in the case of $T = T_{\mathsf{LIP}}$ and $d = 1$.

The preceding analysis indicates that those users who use transforms in various classes (T_{ED}, T_{EDM}, T_{LI}, T_{LIP}) tend to do so again. It is tempting to assume that this is the case even for smaller (but still natural) classes of transforms, such as the component subclasses of T_{LI}, namely T_{cap}, T_{del}, T_{dup}, T_{sub}, T_{ins}, T_{leet}, T_{mov}, and T_{key}. To our surprise, however, we found this is *not* always the case. Specifically, Figure 10(a) shows $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ for experiments using the same methodology as used in Figure 9 (i.e., filtering using $\mathcal{T}_{\mathsf{past}}$), but with $d = 3$ and with T_{past} set to one or the union of two component subclasses of T_{LI}. (In Figure 10, we define T_{past} using $T_{\mathsf{past}} = T^1_{\mathsf{past}} \cup T^2_{\mathsf{past}}$, where T^1_{past} and T^2_{past} range over $\{T_{\mathsf{cap}}, T_{\mathsf{del}}, T_{\mathsf{dup}}, T_{\mathsf{sub}}, T_{\mathsf{ins}}, T_{\mathsf{leet}}, T_{\mathsf{mov}}, T_{\mathsf{key}}\}$.) Note that in Figure 10(a), where $T = T_{\mathsf{past}}$, most values for $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r))$ are relatively small in comparison to those in Figure 9. In fact, of the subclasses of T_{LI}, only the previous use of a transform from T_{sub} is a very strong predictor for the future use of such a transform again. In contrast, many other classes are good predictors for the future use of transforms in T_{LI} in general, as shown in Figure 10(b) where $T = T_{\mathsf{LI}}$. As an interesting and extreme example, the previous use of a transform in $T_{\mathsf{leet}} \cup T_{\mathsf{key}}$ is not a good predictor for the future use of a transform from that class ($\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r)) = .05$ in Figure 10(a)) but is a very strong predictor for the future use of some transform from T_{LI} in general ($\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r)) = .75$ in Figure 10(b)).

5. DISCUSSION

It is possible that some will view our study as motivation to employ transform-based proactive password checking for new passwords as they are chosen, and indeed our transform-based algorithm could be used to implement such a proactive password checker. We caution against this, however. It would not be straightforward

T^1_{past} \ T^2_{past}	T_{cap}	T_{del}	T_{dup}	T_{sub}	T_{ins}	T_{leet}	T_{mov}	T_{key}
T_{cap}	.11	.10	.18	.62	.08	.05	.11	.12
T_{del}		.12	.20	.62	.27	.06	.13	.11
T_{dup}			.21	.62	.07	.13	.14	.17
T_{sub}				.62	.62	.60	.61	.61
T_{ins}					.05	.04	.09	.07
T_{leet}						.04	.10	.05
T_{mov}							.09	.09
T_{key}								.13

(a) $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r)), T = T_{\text{past}}$

T^1_{past} \ T^2_{past}	T_{cap}	T_{del}	T_{dup}	T_{sub}	T_{ins}	T_{leet}	T_{mov}	T_{key}
T_{cap}	.37	.50	.54	.71	.53	.53	.52	.50
T_{del}		.60	.63	.71	.77	.57	.55	.61
T_{dup}			.59	.70	.69	.58	.55	.60
T_{sub}				.72	.71	.70	.68	.72
T_{ins}					.66	.60	.55	.64
T_{leet}						.59	.54	.75
T_{mov}							.54	.55
T_{key}								.63

(b) $\mathbb{P}(\pi_r \in \mathcal{T}(\sigma_r)), T = T_{\text{LI}}$

Figure 10: Evaluation of accounts with history of transform use (§4.2), $T_{\text{past}} = T^1_{\text{past}} \cup T^2_{\text{past}}$, $d = 3$. Averages of 10 trials.

to explain to a user the new passwords she must avoid (or why her chosen password is unacceptable), thereby compounding the already considerable frustration that users already experience due to password expiration (e.g., [1, 17]). For example, the most effective instance of our transform-based framework that we report here involves applying 534 transforms at a depth of three levels; were this used as a proactive password checker, explaining the unacceptable passwords to users would be a challenge, to say the least.

We believe our study casts doubt on the utility of forced password expiration. Even our relatively modest study suggests that at least 41% of passwords can be broken offline from previous passwords for the same accounts in a matter of seconds, and five online password guesses in expectation suffices to break 17% of accounts. As we expand our consideration to other types of transform trees, we would not be surprised to see these success rates jump significantly. Combined with the annoyance that expiration causes users, our evidence suggests it may be appropriate to do away with password expiration altogether, perhaps as a concession while requiring users to invest the effort to select a significantly stronger password than they would otherwise (e.g., a much longer passphrase).

In the longer term, we believe our study supports the conclusion that simple password-based authentication should be abandoned outright. There is already considerable evidence that human-chosen passwords are typically too weak to survive a patient brute-force attacker; see the related works discussed in §2, not to mention our own password cracking activity described in §4 to support our study. The alternatives are well-known: biometrics, device-based solutions, etc. While debating these alternatives is outside the scope of the present paper, we believe that many should be preferable to the status quo, and will only become more so as the imbalance between attacker resources and user memory grows.

6. CONCLUSION

Password expiration is widely practiced, owing to the potential it holds for revoking attackers' access to accounts for which they have learned (or broken) the passwords. In this paper we present the first large-scale measurement (we are aware of) of the extent to which this potential is realized in practice. Our study is grounded in a novel search framework and an algorithm for devising a search strategy that is approximately optimal. Using this framework, we confirm previous conjectures that the effectiveness of expiration in meeting its intended goal is weak. Our study goes beyond this, however, in evaluating susceptibility of accounts to our search techniques even when passwords in those accounts are individually strong, and the extent to which use of particular types of transforms predicts the transforms the same user might employ in the future. We believe our study calls into question the continued use of expiration and, in the longer term, provides one more piece of evidence to facilitate a move away from passwords altogether.

Acknowledgements.

We thank Alex Everett and Karsten Honeycut for facilitating access to the data used in this study; Anupam Gupta for helpful discussions on approximation algorithms; and the anonymous reviewers for their comments. This work was supported in part by NSF grants 0756998 and 0910483.

7. REFERENCES

[1] A. Adams and M. A. Sasse. Users are not the enemy. *Comm. ACM*, 42(12):40–46, December 1999.

[2] S. Alexander, Jr. In defense of password expiration. Post to LOPSA blog, April 2006. `http://lopsa.org/node/295` as of March 28, 2010.

[3] S. M. Bellovin. Unconventional wisdom. *IEEE Security & Privacy*, 4(1), January–February 2006.

[4] M. Bishop. Proactive password checking. In *In Proc. 4th Workshop on Computer Security Incident Handling*, 1992.

[5] S. L. Brand and J. Makey. Department of Defense password management guideline. CSC-STD-002-85, Department of Defense Computer Security Center, April 1985.

[6] W. E. Burr, D. F. Dodson, W. T. Polk, P. J. Bond, and A. L. Bement. NIST special publication 800-63, version 1.0.1, 2004.

[7] S. Chiasson, A. Forget, E. Stobert, P. C. van Oorschot, and R. Biddle. Multiple password interference in text passwords and click-based graphical passwords. In *16th ACM Conference on Computer and Communications Security*, pages 500–511, November 2009.

[8] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3:1–19, February 2007.

[9] A. M. De Alvare. How crackers crack passwords or what passwords to avoid. In *Second Security Workshop Program*, pages 103–112. USENIX, August 1990.

[10] A. M. De Alvare and E. E. Schultz, Jr. A framework for password selection. In *UNIX Security Workshop Proceedings*, pages 8–9. USENIX, August 1988.

[11] U. Feige, L. Lovász, and P. Tetali. Approximating min sum set cover. *Algorithmica*, 40:219–234, 2004.

[12] D. Florêncio and C. Herley. A large-scale study of web password habits. In *WWW 2007*, May 2007.

[13] A. Forget, S. Chiasson, P. C. van Oorschot, and R. Biddle. Improving text passwords through persuasion. In *SOUPS*, 2008.

[14] D. V. Klein. Foiling the cracker: A survey of, and improvements to, password security. In *Second Security Workshop Program*, pages 5–14. USENIX, August 1990.

[15] C. Kuo, S. Romanosky, and L. F. Cranor. Human selection of mnemonic phrase-based passwords. In *SOUPS*, pages 67–78, July 2006.

[16] B. Lu and M. B. Twidale. Managing multiple passwords and multiple logins: MiFA minimal-feedback hints for remote authentication. In *IFIP INTERACT 2003 Conference*, pages 821–824, 2003.

[17] N. Massad and J. Beachboard. A taxonomy of service failures in electronic retailing. In *41st Hawaii International Conference on System Sciences*, 2008.

[18] R. Morris and K. Thompson. Password security: A case history. *Comm. ACM*, 22:594–597, November 1979.

[19] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *12th ACM Conference on Computer and Communications Security*, pages 364–372, November 2005.

[20] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology – CRYPTO 2003*, pages 617–630, August 2003.

[21] B. Patterson. Letter to *Comm. ACM*. *Comm. ACM*, 43(4):11–12, April 2000.

[22] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor. Encountering stronger password requirements: User attitudes and behaviors. In *6th Symposium on Usable Privacy and Security*, July 2010.

[23] E. H. Spafford. Opus: Preventing weak password choices. *Computers & Security*, 11:273–278, 1991.

[24] G. Spafford. Security myths and passwords. Post to CERIAS blog, April 2006. `http://www.cerias.purdue.edu/site/blog/post/password-change-myths/` as of March 28, 2010.

[25] J. M. Stanton, K. R. Stam, P. Mastrangelo, and J. Jolton. Analysis of end user security behaviors. *Computers & Security*, 24(2):124–133, 2005.

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

[27] J. Yan. A note on proactive password checking. In *ACM New Security Paradigms Workshop*, pages 127–135, 2001.

[28] J. Zhang, X. Luo, S. Akkaladevi, and J. Ziegelmayer. Improving multiple-password recall: an empirical study. *European Journal of Information Systems*, pages 1–12, 2009.

APPENDIX

A. TRANSFORM SETS T_{LI} AND T_{LIP}

In this appendix, we elaborate on two of the sets of primitive transforms used in our experiments, namely T_{LI} and T_{LIP}. T_{LIP} is comprised of the top 50 transforms in T_{LI}. These 50 transforms are listed in Figure 11 in Perl regular expression syntax; e.g., `s/1/2/` replaces the first occurrence of "1" with "2". Also listed for each transform t in the column labeled $\mathbb{P}(t(\sigma_r) = \pi_r)$ is the probability, under random choice r of account, that t $(\sigma_r) = \pi_r$, averaged over 1000 trials in which $\sigma_{1..n}$ and $\pi_{1..n}$ are populated as in §4.1. This is the criterion by which these transforms were selected for inclusion in T_{LIP}, and so all other transforms in T_{LI} succeeded on fewer password pairs than these 50.

The complete set of transforms in T_{LI} is shown in Figure 12. T_{LI} is composed of transforms from eight nonoverlapping sets, denoted T_{cap}, T_{del}, T_{dup}, T_{sub}, T_{ins}, T_{leet}, T_{mov}, and T_{key}. The transform sets T_{del}, T_{sub} and T_{dup} are motivated by the hypothesis that users simply delete, substitute, or repeat one or more characters in the non-alphabetic part of their old passwords when creating new ones. These transform sets capture such behaviors, though the special character substitutions, duplications or deletions they encompass are those involving only the characters of which the password is required to contain one or more (see §4). T_{ins} includes inserting a new number following an existing number, and of value either one more or one less than that existing number. Capitalization (or removing capitalization) of alphabetic characters is captured in T_{cap}. T_{key} contains substitutions that replace a character by the character sharing the same keyboard key, and is restricted to only the numeric keys. T_{leet} contains only the single-character leet transforms shown at `http://qntm.org/l33t` at the time of this writing. Finally, T_{mov} contains substring moves in which two adjacent substrings of the password, one or both consisting of only one "type" of character, are swapped. As these descriptions reveal, many of these transform sets are not exhaustive, and so in this respect, the pessimistic results of this paper are conservative.

Transform t	$\mathbb{P}(t(\sigma_r) = \pi_r)$
`s/1/2/`	0.0126
`s/2/3/`	0.0096
`s/3/4/`	0.0085
`s/4/5/`	0.0079
`s/!/@/`	0.0067
`s/5/6/`	0.0062
`s/6/7/`	0.0056
`s/@/#/`	0.0051
`s/#/$/`	0.0050
`s/7/8/`	0.0050
`s/1/3/`	0.0046
`s/\$/%/`	0.0045
`s/8/9/`	0.0045
`s/2/4/`	0.0044
`s/!/!!/`	0.0044
`s/3/5/`	0.0038
`s/!/#/`	0.0034
`s/4/6/`	0.0030
`s/@/$/`	0.0028
`s/(\W+)(.+)/$2$1/`	0.0027
`s/!/$/`	0.0026
`s/2/5/`	0.0025
`s/6/8/`	0.0025
`s/5/7/`	0.0025
`s/0/1/`	0.0025
`s/#/%/`	0.0025
`s/%/!/`	0.0023
`s/1/4/`	0.0023
`s/7/9/`	0.0021
`s/\*/!/`	0.0020
`s/([a-zA-Z]+)(.+)/$2$1/`	0.0019
`s/&/*/`	0.0019
`s/5/1/`	0.0019
`s/%/&/`	0.0019
`s/9/1/`	0.0018
`s/\$/!/`	0.0018
`s/!/*/`	0.0017
`s/(\D+)(\d+)/$2$1/`	0.0017
`s/%/*/`	0.0016
`s/#/!/`	0.0016
`s/(\d)(\W)/$2$1/`	0.0016
`s/4/1/`	0.0016
`s/3/1/`	0.0015
`s/@/!/`	0.0015
`s/2/1/`	0.0015
`s/1/5/`	0.0015
`s/\*/@/`	0.0014
`s/5/8/`	0.0014
`s/@/%/`	0.0014
`s/3/6/`	0.0014

Figure 11: Composition of T_{LIP}, Perl regular expression syntax

T_{ins}	s/1/12/	s/2/23/	s/3/34/	s/4/45/	s/5/56/	s/6/67/	s/7/78/	s/8/89/	s/9/90/	s/0/01/
	s/1/10/	s/2/21/	s/3/32/	s/4/43/	s/5/54/	s/6/65/	s/7/76/	s/8/87/	s/9/98/	s/0/09/
T_{cap}	s/a/A/	s/b/B/	s/c/C/	s/d/D/	s/e/E/	s/f/F/	s/g/G/	s/h/H/	s/i/I/	s/j/J/
	s/k/K/	s/l/L/	s/m/M/	s/n/N/	s/o/O/	s/p/P/	s/q/Q/	s/r/R/	s/s/S/	s/t/T/
	s/u/U/	s/v/V/	s/w/W/	s/x/X/	s/y/Y/	s/z/Z/	s/A/a/	s/B/b/	s/C/c/	s/D/d/
	s/E/e/	s/F/f/	s/G/g/	s/H/h/	s/I/i/	s/J/j/	s/K/k/	s/L/l/	s/M/m/	s/N/n/
	s/O/o/	s/P/p/	s/Q/q/	s/R/r/	s/S/s/	s/T/t/	s/U/u/	s/V/v/	s/W/w/	s/X/x/
	s/Y/y/	s/Z/z/								
T_{del}	s/1//	s/2//	s/3//	s/4//	s/5//	s/6//	s/7//	s/8//	s/9//	s/0//
	s/!//	s/@//	s/#//	s/\$//	s/%//	s/&//	s/*//	s/\+//	s/=//	s/\{//
	s/}//	s/\?//	s/<//	s/>//	s/"//	s/'//				
T_{dup}	s/1/11/	s/2/22/	s/3/33/	s/4/44/	s/5/55/	s/6/66/	s/7/77/	s/8/88/	s/9/99/	s/0/00/
	s/!/!!/	s/@/@@/	s/#/##/	s/\$/\$$/	s/%/%%/	s/&/&&/	s/*/**/	s/\+/++/	s/=/==/	s/\{/{{/
	s/}/}}/	s/\?/??/	s/</<</	s/>/>>/	s/"/""/	s/'/''/				
T_{key}	s/1/!/	s/2/@/	s/3/#/	s/4/$/	s/5/%/	s/6/^/	s/7/&/	s/8/*/	s/9/(/	s/0/)/
	s/!/1/	s/@/2/	s/#/3/	s/\$/4/	s/%/5/	s/\^/6/	s/&/7/	s/*/8/	s/\(/9/	s/\)/0/
T_{leet}	s/a/4/i	s/a/@/i	s/b/8/i	s/c/(/i	s/c/</i	s/c/k/i	s/c/s/i	s/e/3/i	s/g/(/i	s/g/9/i
	s/g/6/i	s/h/#/i	s/i/1/i	s/i/1/i	s/i/|/i	s/i/!/i	s/l/|/i	s/l/1/i	s/o/0/i	s/s/5/i
	s/s/$/i	s/t/+/i	s/t/7/i	s/y/j/i	s/z/2/i	s/4/a/	s/@/a/	s/8/b/	s/\(/c/	s/</c/
	s/k/c/	s/s/c/	s/3/e/	s/\(/g/	s/9/g/	s/6/g/	s/#/h/	s/l/i/	s/1/i/	s/|/i/
	s/!/i/	s/|/l/	s/1/l/	s/0/o/	s/5/s/	s/\$/s/	s/\+/t/	s/7/t/	s/j/y/	s/2/z/
T_{sub}	s/0/1/	s/0/2/	s/0/3/	s/0/4/	s/0/5/	s/0/6/	s/0/7/	s/0/8/	s/0/9/	s/1/0/
	s/1/2/	s/1/3/	s/1/4/	s/1/5/	s/1/6/	s/1/7/	s/1/8/	s/1/9/	s/2/0/	s/2/1/
	s/2/3/	s/2/4/	s/2/5/	s/2/6/	s/2/7/	s/2/8/	s/2/9/	s/3/0/	s/3/1/	s/3/2/
	s/3/4/	s/3/5/	s/3/6/	s/3/7/	s/3/8/	s/3/9/	s/4/0/	s/4/1/	s/4/2/	s/4/3/
	s/4/5/	s/4/6/	s/4/7/	s/4/8/	s/4/9/	s/5/0/	s/5/1/	s/5/2/	s/5/3/	s/5/4/
	s/5/6/	s/5/7/	s/5/8/	s/5/9/	s/6/0/	s/6/1/	s/6/2/	s/6/3/	s/6/4/	s/6/5/
	s/6/7/	s/6/8/	s/6/9/	s/7/0/	s/7/1/	s/7/2/	s/7/3/	s/7/4/	s/7/5/	s/7/6/
	s/7/8/	s/7/9/	s/8/0/	s/8/1/	s/8/2/	s/8/3/	s/8/4/	s/8/5/	s/8/6/	s/8/7/
	s/8/9/	s/9/0/	s/9/1/	s/9/2/	s/9/3/	s/9/4/	s/9/5/	s/9/6/	s/9/7/	s/9/8/
	s/!/@/	s/!/#/	s/!/$/	s/!/%/	s/!/&/	s/!/*/	s/!/+/	s/!/=/	s/!/{/	s/!/}/
	s/!/?/	s/!/</	s/!/>/	s/!/"/	s/!/'/	s/@/!/	s/@/#/	s/@/$/	s/@/%/	s/@/&/
	s/@/*/	s/@/+/	s/@/=/	s/@/{/	s/@/}/	s/@/?/	s/@/</	s/@/>/	s/@/"/	s/@/'/
	s/#/!/	s/#/@/	s/#/$/	s/#/%/	s/#/&/	s/#/*/	s/#/+/	s/#/=/	s/#/{/	s/#/}/
	s/#/?/	s/#/</	s/#/>/	s/#/"/	s/#/'/	s/$/!/	s/\$/@/	s/\$/#/	s/\$/%/	s/\$/&/
	s/\$/*/	s/\$/+/	s/\$/=/	s/\$/{/	s/\$/}/	s/\$/?/	s/\$/</	s/\$/>/	s/\$/"/	s/\$/'/
	s/%/!/	s/%/@/	s/%/#/	s/%/$/	s/%/&/	s/%/*/	s/%/+/	s/%/=/	s/%/{/	s/%/}/
	s/%/?/	s/%/</	s/%/>/	s/%/"/	s/%/'/	s/&/!/	s/&/@/	s/&/#/	s/&/$/	s/&/%/
	s/&/*/	s/&/+/	s/&/=/	s/&/{/	s/&/}/	s/&/?/	s/&/</	s/&/>/	s/&/"/	s/&/'/
	s/*/!/	s/*/@/	s/*/#/	s/*/$/	s/*/%/	s/*/&/	s/*/+/	s/*/=/	s/*/{/	s/*/}/
	s/*/?/	s/*/</	s/*/>/	s/*/"/	s/*/'/	s/\+/!/	s/\+/@/	s/\+/#/	s/\+/$/	s/\+/%/
	s/\+/&/	s/\+/*/	s/\+/=/	s/\+/{/	s/\+/}/	s/\+/?/	s/\+/</	s/\+/>/	s/\+/"/	s/\+/'/
	s/=/!/	s/=/@/	s/=/#/	s/=/$/	s/=/%/	s/=/&/	s/=/*/	s/=/+/	s/=/{/	s/=/}/
	s/=/?/	s/=/</	s/=/>/	s/=/"/	s/=/'/	s/\{/!/	s/\{/@/	s/\{/#/	s/\{/$/	s/\{/%/
	s/\{/&/	s/\{/*/	s/\{/+/	s/\{/=/	s/\{/}/	s/\{/?/	s/\{/</	s/\{/>/	s/\{/"/	s/\{/'/
	s/\}/!/	s/\}/@/	s/\}/#/	s/\}/$/	s/\}/%/	s/\}/&/	s/\}/*/	s/\}/+/	s/\}/=/	s/\}/{/
	s/\}/?/	s/\}/</	s/\}/>/	s/\}/"/	s/\}/'/	s/\?/!/	s/\?/@/	s/\?/#/	s/\?/$/	s/\?/%/
	s/\?/&/	s/\?/*/	s/\?/+/	s/\?/=/	s/\?/{/	s/\?/}/	s/\?/</	s/\?/>/	s/\?/"/	s/\?/'/
	s/</!/	s/</@/	s/</#/	s/</$/	s/</%/S	s/</&/	s/</*/	s/</+/	s/</=/	s/</{/
	s/<//}/	s/</?/	s/</>/	s/<"/	s/</'/	s/>/!/	s/>/@/	s/>/#/	s/>/$/	s/>/%/
	s/>/&/	s/>/*/	s/>/+/	s/>/=/	s/>/{/	s/>/}/	s/>/?/	s/>/</	s/>/"/	s/>/'/
	s/"/!/	s/"/@/	s/"/#/	s/"/$/	s/"/%/	s/"/&/	s/"/*/	s/"/+/	s/"/=/	s/"/{/
	s/"/}/	s/"/?/	s/"/</	s/"/>/	s/"/'/	s/'/!/	s/'/@/	s/'/#/	s/'/$/	s/'/%/
	s/'/&/	s/'/*/	s/'/+/	s/'/=/	s/'/{/	s/'/}/	s/'/?/	s/'/</	s/'/>/	s/'/"/
T_{mov}	s/(\d+)(.+)/$2$1/		s/(\D+)(\d+)/$2$1/			s/(\W+)(.+)/$2$1/		s/(\w+)(\W+)/$2$1/		
	s/(\d+)(\d+)/$2$1/		s/(\W+)(\W+)/$2$1/			s/(\W+)(\d+)/$2$1/		s/(\d+)(\W+)/$2$1/		
	s/([a-zA-Z]+)(.+)/$2$1/					s/([^a-zA-Z]+)([a-zA-Z]+)/$2$1/				

Figure 12: Composition of T_{LI}, Perl regular expression syntax