Chapter 16

USING PERSONAL INFORMATION IN TARGETED GRAMMAR-BASED PROBABILISTIC PASSWORD ATTACKS

Shiva Houshmand and Sudhir Aggarwal

Abstract Passwords are the primary means of authentication and security for online accounts and are commonly used to encrypt files and disks. This research demonstrates how personal information about users can be added systematically to enhance password cracking. Specifically, a dictionary-based probabilistic context-free grammar approach is proposed that effectively incorporates personal information about a targeted user into component grammars and dictionaries used for password cracking. The component grammars model various types of personal information such as family names and dates, previous password information and possible information about sequential passwords. A mathematical model for merging multiple grammars that combines the characteristics of the component grammars is presented. The resulting merged target grammar, which is also merged with a standard grammar, is used along with various dictionaries to generate guesses that quickly match target passwords. The experimental results demonstrate that the approach significantly improves password cracking performance.

Keywords: Password cracking, context-free grammars, personal information

1. Introduction

The use of passwords for authentication is almost universal and the quality of passwords continues to be important despite mechanisms such as two-factor authentication and biometrics. A recent survey [15] reports that a typical user has around 26 online accounts, but only five different passwords. To ensure quality passwords, policies are used by many websites to ensure that users employ strong passwords; however, there are no clear indications about the efficacy of these policies.

G. Peterson and S. Shenoi (Eds.): Advances in Digital Forensics XIII, IFIP AICT 511, pp. 285–303, 2017.
DOI: 10.1007/978-3-319-67208-3_16

Research has demonstrated that many passwords from revealed datasets [13] can be broken via improved password cracking techniques. However, in a real-world situation, a substantial number of passwords are not easily broken because users may employ different forms of passwords for "high value" sites.

During forensic investigations of seized hard drives, law enforcement professionals frequently encounter encrypted volumes or disks. Encryption techniques such as the one used by TrueCrypt are specifically designed to take a long time to compute the hashes, rendering it impractical to try millions of guesses; in such situations, additional resources – beyond more computing power – must be brought to bear.

A promising solution is to leverage personal information about users in cracking their passwords. This research explores how knowledge about a user (target) can be used to enhance the ability of law enforcement to attack passwords belonging to the target. The attacker might leverage specific information about the target such as names of family members, important dates, addresses and numbers, as well as possibly some of the target's previous passwords for the same account or from sister accounts. The approach employs the dictionary-based probabilistic context-free grammar (PCFG) approach to password cracking [6, 16, 17], which trains a grammar on revealed password sets and uses the learned grammar – called the attack grammar – to generate guesses in optimal probability order. Personal information about a target can be incorporated in PCFG-based password cracking in a very straightforward manner. The experimental results demonstrate that the approach augmented with information about a target significantly improves password cracking performance.

2. Background and Related Work

A probabilistic context-free grammar is derived via training on a large set of revealed passwords. This grammar is then used to generate guesses in probability order. If no other information is known, this is referred to as an optimal off-line or on-line password attack. Interested readers are referred to [6, 17] for details about PCFG-based password cracking.

PCFG-based password cracking has two major phases (and code components):

- **Training:** In this phase, first-level production rules of the grammar are learned. These rules generate base structures from a start symbol $\mathcal{S}$. A base structure comprises grammar variables that abstract the class and length of password components. For example, class types can be L denoting alphabet strings, D denoting dig-

its and S denoting special characters. Additionally, K is used for keyboard patterns and M for multi-word patterns. Capitalization is represented by the class type U/N (uppercase/lowercase), but they are treated slightly differently. The length of a substring is indicated as a part of a variable. For example, $\mathcal{S} \rightarrow M_8 D_4 S_2$ is a rule that derives a base structure of three variables of the indicated lengths.

Second-level grammar rules derive terminal substrings from base structure variables such as $D_4 \rightarrow \textit{2001}$. For example, the derivation of the password *iloveyou2001##* is:

$$\begin{aligned} \mathcal{S} &\Rightarrow M_8 D_4 S_2 \\ &\Rightarrow \textit{iloveyou} D_4 S_2 \\ &\Rightarrow \textit{iloveyou2001} S_2 \\ &\Rightarrow \textit{iloveyou2001\#\#} \end{aligned}$$

The probabilities of most of these secondary rules are also learned from the training set. Probability smoothing can be used to give appropriate probability values to rules that are not found in the training set. To calculate the probability of a string derived from the start symbol, the probabilities of the rules used in the various steps are multiplied together. Additional information used for cracking is incorporated in dictionaries, which contain words that can replace the L and M structures when generating guesses.

- **Guess Generation:** In this phase, the learned probabilistic context-free grammar and a set of multiple (attack) dictionaries are used to generate password guesses in decreasing probability order. Probabilities are associated with each dictionary. A dictionary can be generated by learning from the training set or via other means. The probabilities of the actual words used during guess generation depend on the dictionaries and their probabilities.

Little research has focused on using personal information, including previous passwords, to improve password cracking. One reason is that very few revealed password datasets with personal information were available. A second reason is ethical concerns about using email and personal information about users, even if they were leaked by hackers and posted on public websites. For example, if one has a revealed email and password, the ethical dilemma is whether it is appropriate to look online to find additional personal information about the user even if the new information is not made public. Several authors [1, 7, 14] have

recently used leaked information to explore how users create passwords using personal information and how this information may be used to crack passwords.

Castelluccia et al. [1] have attempted to leverage personal information to improve password cracking using a Markov model [10]. Their OMEN+ system integrates personal information with normal training based on revealed passwords. The fundamental difficulty with the Markov approach [4, 10] is that it is computationally expensive to generate passwords in probability order because the training typically uses the learned probabilities of n-grams to determine the probability of a password string. Thus, it is first necessary to create various discretized probability levels assigned to buckets in which strings are placed based on the 3-gram probabilities of strings in the training set. The strings in the buckets are attempted in the highest probability order of the buckets.

Castelluccia et al. [1] have also explored the use of personal information such as email, birthdays and usernames, but did not use previous passwords or cross-site passwords. Their OMEN+ system attempts to determine if a password set containing personal information has any overlap with a password with personal information. If an overlap exists, then the corresponding probabilities of the overlapping 3-grams are increased based on a parameter. For some reason, instead of targeting a specific individual, Castelluccia et al. modify the 3-gram probabilities such that better results can be obtained for a complete test set. In contrast, the proposed approach uses a highly efficient PCFG-based training system without the problems of the Markov approach. Moreover, it does not require changes to the training or cracking components; instead, only additional grammars and dictionaries have to be created.

Li et al. [7] have explored the use of personal information in password cracking using the 12306 dataset that was leaked from a Chinese railway ticket website. The dataset contains approximately 130,000 Chinese passwords as well as personal information such as email address, username, cell phone number and the user's Chinese name. Additional information that can be extracted includes the birthday and gender of a user. Li et al. extended the probabilistic context-free grammar approach of Weir et al. [17] to develop the Personal-PCFG system. The extension adds a new grammar variable for each type of personal information such as B for birthday, N for name and E for email address; additionally, as in other probabilistic context-free grammars, subscripts are used to indicate lengths. During the preprocessing phase, a password such as *helloalice816!* is converted to the structure $helloN_5B_3!$ if the personal information (name and birthday) match.

The problem with this approach is that Li et al. do not appreciate the grammar ambiguity that arises when several base structures yield the same terminal string. Generating guesses in the order of highest probability when using a context-free grammar relies on the grammar being unambiguous. This ensures that there is a well-defined probability for a guess that depends on a single unique derivation.

The ambiguity problem is discussed by Houshmand et al. [6], where several new variables such as K for keyboard combinations and M for multiword combinations are introduced to minimize or eliminate grammar ambiguity. Note that the training and cracking components have to be modified to accommodate the new variables. Furthermore, the highest probability order of guesses generated cannot be maintained unless an effort is made during training to eliminate ambiguity. Otherwise, the PCFG-based approach would still generate guesses for all the base structures, but the probability order of guesses would not be preserved.

Wang et al. [14] also extend probabilistic context-free grammars [17] to accommodate personal information and explore the use of previous passwords. They follow the approach of Li et al. [7], except that instead of using a new variable B for personal information such as birthday with the subscript indicating length, they incorporate a tagged variable system where specific subscripted variables reflect different formats for a birthday. Thus, for each type of personal information, they have to predefine different formats for the particular variable; these become the only formats that are learned during the training process.

The approach of Wang et al. has exactly the same, if not worse, ambiguity problems as that of Li et al. when generating guesses because many different base structures can yield identical passwords, leading to incorrect assignments of probabilities to the guesses. For example, the password string *120982* can be derived from various variables represented as B_1, B_2, ... , B_{10} or even D_6, which would then give incorrect probabilities to terminal strings during guessing because all possible probabilities of the string derivations have to be added. When using previous password information, Wang et al. also introduce new variables to the grammar that represent various transformations of passwords. But this causes similar problems as discussed above. In contrast, the approach for accommodating previous passwords presented in this chapter does not require changes to probabilistic context-free grammar training or guess generation.

Das et al. [3] have used publicly-available leaked password sets with user identifiers and have analyzed the data to find passwords for the users. In particular, they were able to find 6,077 unique users with at most two passwords for each user; 43% were identical passwords and the

remaining were non-identical. However, Das et al. do not consider the changes that a user might make when using similar passwords for the same or other accounts.

Zhang et al. [18] have conducted a large-scale study focusing on password changes necessitated by password expiration. They were able to access a dataset of more than 7,700 accounts containing a known password and a subsequently changed password for each account. They model a password change as a sequence of transforms (based on various criteria) and organize the transforms as a tree with the old password as the root. A path in the tree is a sequence of transforms that yields the new password with common subsequences being the same from the root. A search starts at the root with an input password and, upon visiting each node in the tree, the corresponding transform is applied to the output of the parent node. Then, each output is tested as a password guess against the target password hash. The primary limitation of this algorithm is its time complexity; thus, the depth of the tree is restricted to three levels. Furthermore, the algorithm does not incorporate information about a user. In contrast, the approach proposed in this chapter derives a new grammar that precisely incorporates user information to enhance PCFG-based password cracking. The approach enables the use of specific information about a previous password instead of generic transformations that a number of users may employ. Additionally, the approach can simultaneously incorporate personal information about the targeted user.

3. Building a Targeted Attack

This section discusses the proposed approach for creating a probabilistic context-free grammar designed to crack a password for a targeted user. It is assumed that there is a single target and that some personal information about the target and the password hash are available. The goal is to create an attack grammar that specifically generates guesses for the target. Note that multiple hashes are not targeted simultaneously. This is not really a limitation because, if the hashes are salted, attacks have to be re-executed for each hash. Also, this is a common situation that holds true for online attacks on the target; this is because there is no notion of trying multiple users simultaneously. If no personal information is available, the same grammar would most likely be used for all targets. Of course, some general information could guide the use of different training datasets (e.g., Chinese vs. English targets). The added complexity of the proposed approach is simply that a grammar is developed for each target. As will be discussed later, grammars can

be combined to handle a situation where the personal information was used to create the password as well as a situation where it was not (e.g., the user simply created some other password).

The available personal information can be as simple as the username or the first and last names of the user or it can be more detailed such as the names of family members, their dates of birth and addresses. Password policies often do not allow users to use their login ids or even their real names when creating passwords. However, human beings tend to use phrases, names and numbers that are familiar to them for easy memorization. Thus, personal information gathered about a target can be very useful in password cracking. The proposed approach can leverage any type of personal information about a target with only minor categorization.

The next section discusses the combination of multiple grammars to create a new grammar that models the use of the component grammars in a precise way. Following this, the use of the combined grammar in cracking passwords belonging to a specific target is described.

3.1 Merging Context-Free Grammars

Generating a probabilistic context-free grammar from a training set of disclosed user passwords can be time consuming depending on the size of the training set. Merging two or more grammars gives the advantage of combining two training sets without having to repeat the training phase.

Suppose it is desired to concatenate two training sets to create a grammar. One way is to merge the training sets and produce a context-free grammar from the entire set. On the other hand, assume that two grammars have been generated, each using a different training set. Then, the two grammars can be merged to create a new grammar that is the result of training using both the sets. This technique also permits the specification of a weight for each grammar to control how much it is affected by each training set.

A probabilistic context free grammar has a set of production rules R_j ($j = 1..n$) where n is the number of rules. Each rule has a single variable (or non-terminal) on the left-hand side with a sequence of variables or terminals on the right-hand side. Each rule R_j has an associated probability p_j with the requirement that the probabilities of all the rules with the same left-hand side must sum to one.

Definition: Let G_1 and G_2 be two probabilistic context-free grammars with base structures and component structures as defined in [6, 17].

Then, the new grammar G, which is called the "merge" of G_1 and G_2, is given by:

$$G = \alpha G_1 + (1 - \alpha)G_2 \qquad where \quad 0 \leq \alpha \leq 1$$

Note that this is only representational because grammars are actually complex abstract tuples. It is assumed that the variables and terminals in the two grammars are chosen from the same possible sets. The parameter α is used to give an appropriate weight to grammar G_1 versus grammar G_2.

Next, it is necessary to define the new set of rules and their probabilities for the merged grammar G.

Definition: Let R be a grammar rule that is in G_1 or G_2. Let the probability of R in G_1 be p_1 and the probability of R in G_2 be p_2 (if R is not in a grammar, then its probability is zero). Then, the probability p of R in the merged grammar G is given by:

$$p = \alpha p_1 + (1 - \alpha)p_2$$

It is easily shown that G is a well-defined probabilistic context-free grammar. This is because the probability values of all rules with the same left-hand side variable in the merged grammar sum to one. Furthermore, the combination of the two grammars can be viewed as an affine transformation with the points being grammars in an abstract space. Intuitively, it is possible to combine any number of grammars by simply ensuring that the sum of their component weights is equal to one and the resulting grammar is the same regardless of the ordering of the combinations.

Table 1 shows a simple example involving the merging of two grammars. The following sections discuss how merged grammars can be used to crack passwords belonging to a target.

3.2 Integrating Personal Information

The approach enables a user to input almost any available personal information about a target. Law enforcement personnel often encounter cases in which they have to break the passwords of suspects about whom they have significant personal information. Examples include the names of family members and friends, usernames, relevant numbers (social security and phone numbers), addresses (street name, city, state and zip code), important dates, favorite sports teams and players. Personal information is entered in a structured way such that the entries can be massaged as needed.

Table 1. Merging grammars G_1 and G_2 with $\alpha = 0.8$.

Grammar G_1	
Rule	**Probability**
$\mathcal{S} \rightarrow L_5D_3 \mid L_5S_1$	0.6 | 0.4
$D_3 \rightarrow$ *999* | *124*	0.8 | 0.2
$S_1 \rightarrow$ *!* | *@*	0.63 | 0.37
Grammar G_2	
Rule	**Probability**
$\mathcal{S} \rightarrow L_5S_1 \mid L_4S_1$	0.7 | 0.3
$D_3 \rightarrow$ *123* | *124*	0.6 | 0.4
$S_1 \rightarrow \#$ | *&*	0.72 | 0.28
Merged Grammar	
Rule	**Probability**
$\mathcal{S} \rightarrow L_5D_3 \mid L_5S_1 \mid L_4S_1$	0.48 | 0.46 | 0.06
$D_3 \rightarrow$ *999* | *124* | *123*	0.64 | 0.24 | 0.12
$S_1 \rightarrow$ *!* | *@* | $\#$ | *&*	0.504 | 0.296 | 0.144 | 0.056

Based on the personal information (PI), a PI-grammar and PI-dictionary are constructed for password cracking. Numbers are added to the digit variables of the PI-grammar. Names, words and alphabet character strings are added directly to the PI-dictionary. Multiple dictionaries can be used during the attack phase (guess generation). For example, a basic dictionary would be a fairly large standard dictionary of words; these words are not learned via the training process because the training set would be too sparse to accurately account for or reflect word usage. Additional specialized dictionaries could be used; examples include a "top words" dictionary that contains the most frequently-occurring words in the training set and the PI-dictionary that contains words based on personal information for use in a targeted attack. The use of the PI-dictionary during guess generation ensures that the strings in the dictionary will be used with higher probabilities in the guesses.

Dates are broken down into month, day and year components, and most variations of dates are similarly added to the PI-grammar. For example, when *02/10/2016* is entered, the following numbers are added to the digit components of the PI-grammar: *02*, *10*, *2016*, *16*, *02102016*, *021016*, *10022016*, *100216*, *0210*, *1002*, etc. The name of the month and its variations are also added to the PI-dictionary (e.g., *February* and *Feb*). Note that the PI-grammar alone is not very useful in password cracking because it has very few base structures and components. The

proposed approach merges this grammar with a general grammar in order to generate guesses.

3.3 Using Old Password Information

As the number of accounts per user increases, users are more likely to reuse their passwords or change their passwords slightly to avoid memorizing new passwords. This is particularly true when a security policy forces users to change their passwords frequently. A survey by Shay et al. [12] conducted on 470 university students, staff and faculty revealed that 60% of the individuals used one password with slight changes for different accounts. Moreover, a study of leaked password sets by Durmuth et al. [4] demonstrated that users often apply simple tricks to make slight changes to their old passwords.

Old passwords contain vital information such as important numbers, dates and names of family members. The goal is to specify a grammar that can generate guesses similar to an old password. Since users often change their passwords with slight modifications, the AMP edit distance [5] is used to define a metric for similar passwords and to determine a grammar that can generate such passwords. AMP uses a distance function to create strengthened passwords within an edit distance of one of a user-chosen password based on the Damerau-Levenshtein edit distance [2]. The Damerau-Levenshtein edit distance measures the minimum number of operations needed to transform one string into another. The AMP version of this distance function includes insertion, deletion, substitution and transposition of components of the base structure as well as similar operations within a component. The AMP distance function is improved by adding operations on keyboard patterns and multiword patterns. These patterns were originally incorporated in probabilistic context-free grammars by Houshmand et al. [6]. For multiwords, the revised AMP edit distance has the unit operations:

- **Insertion:** Insert a D_1 or S_1 in between two words in a multiword. For example, for a password containing *starwars* (M_8), *star5wars* or *star!wars* are created.

- **Transposition of Components:** Transpose two adjacent words as well as the first and last word in a multiword. For example, *mysweetbaby* can be changed to *sweetmybaby*, *mybabysweet* and *babysweetmy*.

- **Deletion:** Delete a word from a multiword, which results in a new base structure as well as a new multiword in the grammar. For example, given the password *mysweetbaby12* with base struc-

ture $M_{11}D_2$, it is possible to create other base structures such as M_9D_2, M_6D_2, M_7D_2 as well as *mysweet*, *mybaby* and *sweetbaby* as multiwords in the grammar.

This approach produces an ED-grammar (edit distance grammar) that, by itself, generates all the guesses within a revised AMP edit distance of one from an old password. Note that every possible change is considered to be equally probable.

3.4 Predicting New Passwords

This section assumes that the attacker has even more information about the target. The focus is on the knowledge about the target's password habits and it is assumed that the attacker has access to at least two similar subsequent old passwords. The first approach can be leveraged to use the passwords to generate an ED-grammar. However, it is also possible to gather information about the changes made to the previous passwords of the target and use this information to predict the new password. In the following paragraphs, an algorithm for determining changes between two subsequent known passwords is presented. Following this, a new password based on the information is predicted.

In order to determine the changes between two passwords, a function is implemented that finds the minimum edit distance by creating a distance matrix. The function also incorporates a backtracking algorithm that determines the operations made between the two strings. The edit distance function is based on the Damerau-Levenshtein algorithm. The algorithm starts by filling a (distance) matrix D of size $n_1 \times n_2$, where n_1 is the length of the first string s and n_2 is the length of the second string t. The $D[i,j]$ value measures the distance between the initial substring of s of length i and the initial substring of t of length j. At the time of creating the matrix, the operations associated with each step are captured and stored in another matrix O (using i: insertion, d: deletion, t: transposition). This is used to create the transformation algorithm that backtracks using the matrix and finds the exact operations made when transforming one string to another.

Hierarchical Transformation Algorithm. Note that the AMP edit distance function [5] is different from the regular Damerau-Levenshtein edit distance. The main difference is in the transposition operation. The Damerau-Levenshtein edit distance considers a transposition of two adjacent characters as one edit distance. While this can be useful to model string similarities, it is not appropriate for password changes. For example, when two passwords such as *iloveyou123* and *123iloveyou*

are compared, the Damerau-Levenshtein edit distance between the two strings is computed as 6 (the algorithm finds that there are three insertions in the beginning and three deletions at the end of the string). However, when considering these two strings as passwords with different components, it is clear that the target has only made one change by transposing the multiword component M_8 with the digit component D_3.

This is modeled using a hierarchical transformation algorithm that first finds the edit distance between the simple base structures. A simple base structure is the base structure of the password without considering the length of each component. For example, the simple base structure of *love456!* is *LSD*.

Given two old subsequent passwords, in the first level, both passwords are parsed to their simple base structures. Then, the edit distance algorithm is invoked for these two simple base structures to determine the differences in the base structures. Using the backtracking algorithm, the operations that caused the change in the simple base structure are determined.

The backtracking algorithm starts from the bottom-right corner of the matrix and travels back to the upper-left corner of the matrix and, in each step, determines the operation that was performed to calculate the edit distance. It then creates a string of the operations along the path that shows how one string has been transformed to the other. If a transposition has been made within the simple base structure, the algorithm checks the values of each component and then reverses the transposition such that it neutralizes the initial transposition effect and recreates one of the passwords similar to the other by applying the transposition.

The second level of the hierarchical algorithm proceeds to find the edit distance between the changed password along with the second password to identify the edit distance and the operations between the two strings. For example, the backtracking function returns nndnnnnnt (n: no change, d: deletion, t: transposition) given *123alice!$* and *12alice$!*. This hierarchical transformation algorithm is used in the next section to predict the changes that the target has made to create the new password.

Creating the Grammar. The transformations between two old passwords can be used to generate guesses of the most recent password of the target. Some of the most common and important changes based on the available data and the results of other studies [3, 18] are used to create the PM-grammar (password modifications grammar). The following functions add appropriate structures to the PM-grammar:

- **Increment/Decrement the Digit Component:** An increment or decrement of one in the digit component in the old passwords is

identified. Upon finding such a change, the next predictable change is added to the grammar. For example, if the old passwords are *bluemoon22* and *bluemoon23*, the number *24* is added to the D_2 component of the grammar. The same base structure L_8D_2 also has to be added to the grammar to increase the likelihood of using the same structure again.

- **Insertion of the Same Digit:** Algorithms have been developed to recognize if a digit has been inserted into a password and if it has been added repeatedly. Examples are *bluemoon* → *bluemoon5* → *bluemoon55* → *bluemoon555*. In this case, if the old passwords are *bluemoon3* and *bluemoon33*, *333* is added to D_3 in the grammar as well as L_8D_3 to the base structures of the grammar.
- **Capitalization of Alpha Strings:** If the old passwords both have the same alpha sequence with different capitalizations, both the capitalizations are added to the grammar because the chances of using the masks are higher.

3.5 Merging Grammars and Generating Guesses

After the PI-grammar, ED-grammar and PM-grammar have been constructed based on the kind of information available, the grammars are merged with a more comprehensive general grammar that is used for password cracking when no personal information is known. By assigning appropriate weights to the grammars, the generated guesses can be balanced such that guesses with personal information are typically generated earlier with higher probability values and guesses that are more general are typically generated later in the guessing process.

This approach, unlike other methods [7, 14], does not require changing the training code or the cracking code of the probabilistic context-free grammar that implements training and guessing. In fact, it can be viewed as an add-on intermediate step that requires no changes to training and guessing.

After training on a large password set (general grammar), the PI-grammar, ED-grammar and PM-grammar are created based on the available personal information. These grammars are merged with the general grammar. The resulting final target grammar can be used as before in offline or online attacks and can generate a wide variety of guesses (in highest probability order) while giving higher probabilities to passwords similar to those used by the target.

The advantage is that no matter which version of a probabilistic context-free grammar is used, the approach still holds. Note that the attacker uses personal information in the hope that the target has used

personal information to create the password. Clearly, the merging of grammars is a powerful mechanism.

A key aspect of the merging technique is that, if the target has not used personal information in passwords, the password cracking system will still work appropriately. For example, merging a general grammar and a special targeted grammar based on a seed password can be used to generate passwords that favor the seed password, but it is not stuck with a limited number of guesses and generates guesses based on the general grammar as well. Furthermore, by adjusting the weights when merging the grammars, it is possible to favor one approach (general) versus the other (seed).

4. Experiments

This section discusses the experimental results on the effectiveness of targeted password attacks. It can be proven that, if personal information (excluding previous passwords) was used in a password such as a name or date, the password cracking approach would automatically use such passwords earlier in guess generation because the probabilities of these passwords are higher due to the grammar merging operation. Unfortunately, obtaining a validated personal information password set or simulating one through user studies is problematic. Therefore, no tests were performed on the PI-grammar. Instead, the experiments focused on a situation where previous passwords are available.

The first experiment compared attacks with and without the ED-grammar. These attacks are referred to as targeted and general attacks, respectively. The (extended) probabilistic context-free grammar of Houshmand et al. [6] was used for training. In the general attack, training was performed using a large dataset of real user passwords and the (general) grammar was used to generate guesses. In the targeted attack, the old password was used to create the ED-grammar, which was then combined with the same general grammar produced for the general attack while also using the same dictionary.

A total of 300,000 passwords were randomly selected from the Yahoo! set of real user passwords that was leaked in 2012 [9]. This training set was used to produce the general grammar. The training approach requires a training dictionary to determine multiwords [6]. EOWL [8] augmented with common proper names and top words from movie scripts was used as in [5]. Additionally, dict-0294 [11] was used as the primary attack dictionary. The test set contained 56 pairs of old and new passwords obtained through a survey of how users change passwords. The

survey study was approved by the Florida State University Institutional Review Board (IRB).

4.1 Password Survey

About 2,000 randomly-selected students from Florida State University were invited to participate in the password survey. The participants were asked to create an account with a password on the survey website. The only password policy requirement was to use at least eight characters. The participants were asked to log in once a day for a total of four times during a period of one week. At each login, the participants were asked a few survey questions. On the third website visit, the participants were asked to change their passwords. On the fourth visit, the participants logged in with their changed passwords and completed the survey.

Multiple logins enabled the participants to gain familiarity and become comfortable with the passwords they had created before they were asked to change their passwords. A total of 144 participants created accounts and 56 of them proceeded to change their passwords; 53% of the participants were female and 47% male. About 40% said that they did not create new passwords for the survey, but simply used their old passwords. Only 14% said that they created new passwords for each account. When creating their passwords, 30% of the participants said they modified their existing passwords, about 24% reused their old passwords and only 14% created new passwords. These percentages are consistent with other studies.

4.2 Testing and Cracking Results

After the participants had changed their passwords, they were asked how they changed their passwords and whether they changed them by modifying their previous passwords. This question was important because it revealed which passwords were changed intentionally by modifying the old passwords. Otherwise, it would not have been possible to divine the intentions of the user although the passwords appeared to be similar. Of the 56 passwords, 23 were claimed to be created based on the previous password. Therefore, the analysis only focused on these passwords to check if the passwords could be guessed effectively.

During each cracking session, the old password was used as input to generate the ED-grammar. This grammar was then combined with the general grammar. Guesses were generated for the targeted and general attacks. Table 2 shows the old password used as input, the new password that was attacked and the number of guesses required to find the new password using the targeted and general attacks.

Table 2. Test Result of Targeted Attack.

Old Password	New Password	Guesses in the Targeted Attack	Guesses in the General Attack
tharaborithor	*thorborithara*	–	–
Simba144!	*@Simba2523*	734,505,973	–
$unGl@$$220	*$unGl@$$110*	4,070	–
research!	*Research!*	554	5,059,949,503
starWars@123	*star#Ecit@123*	2,227,558	–
thebigblackdogjumps	*blackdogmoretime*	–	–
Ahk@1453	*Ahk#1453*	12,026	–
qpalzm73	*qpalzm73**	1,810	–
pluto1995	*boonepluto*	–	–
caramba10	*caramba12*	14	11,424,542
Elvis1993!	*Professional1993!2*	–	–
pepper88	*peppergator88*	128,197,109	2,563,504,751
ganxiedajiA1!!	*1ganxiedajiA*	7,794	–
88dolphins!	*55dolphins!*	38,503	–
kannj2013!	*Kannj2013*	97	–
!FSU$qr335	*!FSU$qr335mcddt*	–	–
vballgrl77	*schatzimae*	–	–
nickc1007	*corkn1007*	–	–
sunflower12	*sunflower13*	202	119,336,969
meg51899	*Meg51899**	5,381	–
Research1	*research11*	206	23,728,452
Gleek1993	*Gleek1985*	9,661	1,994,709,669
Oaklea0441	*Oaklea0112*	91,014	–

Since this work seeks to demonstrate that passwords are cracked faster using personal information (e.g., old passwords), the number of guesses was limited to 10 billion in the password cracking session. The idea was to verify that passwords can be cracked in much shorter sessions than in a regular offline password cracking session.

The results in Table 2 reveal that it was possible to guess most of the passwords that were changed slightly. However, a few of the passwords could not be guessed (shown as –) primarily because there were no relevant alpha strings in the dictionaries. For example, the password *tharaborithor* is not in English and the password *vballgrl77* was not actually modified, although the user claimed it was. The results show that only a few of the passwords were broken during a general password cracking attack and it took much longer for the others compared with the targeted attack. Indeed, the targeted attack was more efficient when information about the old passwords of users was incorporated.

To explore the proposed approach further, a small list of 30 sets of previous passwords from a private entity was used. This list contained no other information. All but two of the passwords were pairs and only

two comprised three sequential passwords. The PM-grammar could be used on the two sequential sets and the third password in the sequence could be cracked on the first guess. Furthermore, 78% of the passwords in the list could be cracked, with 66% of the passwords cracked in less than 20 guesses.

5. Conclusions

This research has demonstrated that personal information belonging to targeted users can be systematically incorporated in probabilistic context-free grammars to efficiently generate password guesses. Three grammars, PI-grammar, ED-grammar and PM-grammar, are created based on various pieces of information such as names, dates, numbers and previous passwords. Multiple grammars are merged using a parameter that appropriately weights each grammar and the new merged grammar maintains its probabilistic properties. The proposed approach is an add-on intermediate step between the training and cracking phases of probabilistic context-free grammars, enabling it to be used very easily with all the probabilistic context-free grammar variations.

Experimental results demonstrate that many of the passwords can be guessed using a targeted grammar; however, a general grammar is not as successful. Passwords are also guessed at a much faster rate (many fewer guesses) using a targeted grammar compared with a general grammar. Future research will attempt to evaluate and refine the proposed research by conducting surveys with much larger numbers of participants and, therefore, more password pairs.

References

[1] C. Castelluccia, A. Chaabane, M. Durmuth and D. Perito, When privacy meets security: Leveraging personal information for password cracking, *Computing Research Repository*, abs/1304.6584, 2013.

[2] F. Damerau, A technique for computer detection and correction of spelling errors, *Communications of the ACM*, vol. 7(3), pp. 171–176, 1964.

[3] A. Das, J. Bonneau, M. Caesar, N. Borisov and X. Wang, The tangled web of password reuse, *Proceedings of the Network and Distributed Systems Security Symposium*, 2014.

[4] M. Durmuth, F. Angelstorf, C. Castelluccia, D. Perito and A. Chaabane, OMEN: Faster password guessing using an ordered Markov enumerator, *Proceedings of the Seventh International Symposium on Engineering Secure Software and Systems*, pp. 119–132, 2015.

[5] S. Houshmand and S. Aggarwal, Building better passwords using probabilistic techniques, *Proceedings of the Twenty-Eighth Annual Computer Security Applications Conference*, pp. 109–118, 2012.

[6] S. Houshmand, S. Aggarwal and R. Flood, Next Gen PCFG password cracking, *IEEE Transactions on Information Forensics and Security*, vol. 10(8), pp. 1776–1791, 2015.

[7] Y. Li, H. Wang and K. Sun, A study of personal information in human-chosen passwords and their security implications, *Proceedings of the Thirty-Fifth Annual IEEE International Conference on Computer Communications*, 2016.

[8] K. Loge, The English Open Word List, Dreamsteep (`dreamsteep.com/projects/the-english-open-word-list.html`), 2017.

[9] S. Musil, Hackers post 450K credentials pilfered from Yahoo, *CNET*, July 11, 2012.

[10] A. Narayanan and V. Shmatikov, Fast dictionary attacks on passwords using time-space tradeoff, *Proceedings of the Twelfth ACM Conference on Computer and Communications Security*, pp. 364–372, 2005.

[11] Outpost9.com, Word Lists..., (`www.outpost9.com/files/WordLists.html`), 2004.

[12] R. Shay, S. Komanduri, P. Kelley, P. Leon, M. Mazurek, L. Bauer, N. Christin and L. Cranor, Encountering stronger password requirements: User attitudes and behaviors, *Proceedings of the Sixth Symposium on Usable Privacy and Security*, article no. 2, 2010.

[13] A. Vance, If your password is 123456, just make it hackme, *The New York Times*, January 20, 2010.

[14] D. Wang, Z. Zhang, P. Wang, J. Yan and X. Huang, Targeted online password guessing: An underestimated threat, *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1242–1254, 2016.

[15] R. Waugh, No wonder hackers have it easy: Most of us now have 26 different online accounts – but only five passwords, *Daily Mail*, July 16, 2102.

[16] M. Weir, S. Aggarwal, M. Collins and H. Stern, Testing metrics for password creation policies by attacking large sets of revealed passwords, *Proceedings of the Seventeenth ACM Conference on Computer and Communications Security*, pp. 162–175, 2010.

[17] M. Weir, S. Aggarwal, B. de Medeiros and B. Glodek, Password cracking using probabilistic context-free grammars, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 391–405, 2009.

[18] Y. Zhang, F. Monrose and M. Reiter, The security of modern password expiration: An algorithmic framework and empirical analysis, *Proceedings of the Seventeenth ACM Conference on Computer and Communications Security*, pp. 176–186, 2010.