International Journal of Innovative Computing, Information and Control
Volume **9**, Number **2**, February **2013**

ICIC International ©2013 ISSN 1349-4198
pp. **821–839**

PASSWORD CRACKING BASED ON LEARNED PATTERNS FROM DISCLOSED PASSWORDS

Hsien-Cheng Chou[1], Hung-Chang Lee[2], Hwan-Jeu Yu[1], Fei-Pei Lai[1,3]
Kuo-Hsuan Huang[4] and Chih-Wen Hsueh[1]

[1]Department of Computer Science and Information Engineering
[3]Graduate Institute of Biomedical Electronics and Bioinformatics
National Taiwan University
No. 1, Section 4, Roosevelt Road, Taipei 10617, Taiwan
{ d96922034; flai }@csie.ntu.edu.tw; ecpro@seed.net.tw

[2]Department of Information Management
Tamkang University
No. 151, Yingzhuan Road, Tamsui District, New Taipei City 25137, Taiwan
hclee@mail.im.tku.edu.tw

[4]Department of Computer Science and Engineering
Tatung University
No. 40, Zhongshan North Road, 3rd Section, Taipei 104, Taiwan
khhuang@ttu.edu.tw

Received December 2011; revised April 2012

Abstract. *Password-based authentication systems are still the most commonly used mechanism for protecting sensitive information despite being vulnerable to dictionary based attacks. To guard against such attacks, many organizations enforce complicated password-creation rules and require that passwords include numeric and special characters. This study demonstrates that as long as passwords are not difficult to remember, they remain vulnerable to "smart dictionary" attacks. In this study, a password analysis platform is developed to formally analyze commonly used passwords and identify frequently used password patterns and their associated probabilities. Based upon these patterns, we establish a model consisting of a Training set, a Dictionary set and a Testing set (TDT model) to generate probabilistic passwords sorted in decreasing order. The model can be used to dramatically reduce the size of the password space to be searched. Simulation results show that the number of passwords cracked using the TDT model is 1.43 and 2.5 times higher compared with the John-the-Ripper attack and Brute-force attack, respectively. We also design a hybrid password cracking system combining different attacks to verify the effectiveness of the proposed method. After applying the TDT model, the number of passwords cracked increased by up to 273%.*
Keywords: Password cracking, Dictionary attack, Brute-force attack, TDT model

1. **Introduction.** Identity authentication or access control through passwords is still a widely used method of ensuring system security, despite the increased use of alternative techniques such as graphical passwords [1], smart card, and biometrics. However, password based security is vulnerable to the dictionary attack [2]. In order to force users to create strong passwords, system administration policies often enforce several complex rules intended to force users into creating strong passwords [3,4]. Under such rules, users may be required to use numeric or special characters, have to enter passwords of a minimum length, and avoid words found in a dictionary. Users often struggle to create passwords that meet these requirements. However, it also becomes difficult to crack such passwords using a dictionary attack.

For example, consider the password 'computer123!'. If password rules were not enforced, a user may simply have selected 'computer' as their password. However, in order to satisfy the requirements of the system administration policy, they may have to append numeric characters '123' and a special character '!'. In this way, security is enhanced because the difficulty of cracking the password increases from 26^8 to 68^{12}.

In addition to enforcing high password complexity, many password-based cryptosystems execute multiple iterations in their core encryption algorithm to defend against the Brute-force attack. In other words, they execute the core encryption algorithm many times to increase password complexity and cracking time. Take the WORD software as an example, where an attacker can crack more than 30,000 passwords per second for the 2003 edition using a dual quad-core 2.33GHz Intel computer. However, it is only able to crack around 400 passwords per second for the 2007 edition. Therefore, taking WORD 2007 as an example, for a lowercase string of length 8, it may take 16.5 years $(26^8/400 * 86400 * 365)$ to crack using the Brute-force attack.

These examples suggest that due to rapidly changing user habits for password selection and enhancements in the encryption algorithm, the dictionary attack and brute-force attack face an enormous challenge. This paper, looking from the attacker's standpoint, aims to overcome this problem and design a good method for generating highly efficient passwords for cracking. The proposed method makes the dictionary attack stronger by overcoming its shortcomings and also overcomes the limitations of the brute-force attack.

In this paper, we analyze user habits in creating passwords, and design an effective password attack method. First, we design a password analysis platform using Flex [5,6] with Cygwin [7]. It uses regular expressions to categorize and analyze disclosed passwords to understand how users create passwords. By analyzing the disclosed passwords, we are able to determine how users incorporate the required character types in their passwords and summarize some frequently used password patterns with their associated probabilities. We then create a model comprising a Training set, a Dictionary set and a Testing set, called the TDT model. This model can generate passwords and sort them in decreasing order of probability. Finally, we devise a hybrid password cracking system, which is a three stage sequence comprising a dictionary attack, the TDT model attack, and a brute force attack. We then simulate and apply this system to cracking UNIX access passwords.

This paper is organized as follows. Section 2 describes related research on password attacks. Section 3 proposes pattern analysis from disclosed passwords. Section 4 analyses the strategy for generating passwords. Section 5 presents the simulation results. Section 6 demonstrates the effectiveness and comparisons with other related researches. Section 7 contains conclusion and future work.

2. **Related Work.** Techniques for cracking or acquiring passwords include social engineering, phishing and shoulder surfing. However, most current identity authentication attacks (focusing on passwords) are still based on the dictionary attack or on brute-force methods. Other attacks such as time-memory tradeoff [8,9] are gradually gaining importance as computing power and storage space increases. However, this attack requires substantial computational time and effort to build Rainbow tables [10] and thus has not been widely deployed. In addition, an effective defense has been mounted against Rainbow tables by exploiting the fact that some hash functions are combined with a random salt (for example salted SHA-1), which causes the same passwords to produce different hash values.

Actually, many companies such as Elcomsoft, Passware, and Wwwhack have developed password recovery software for documents files that use character strings for data encryption, such as Word, Excel, PDF, RAR, and ZIP encryption files. However, most of these password recovery packages are still based on the dictionary attack or brute-force attack.

The effectiveness of dictionary attack and brute-force attack has been challenged by the increasing awareness and emphasis on secure password policies. In addition, many password-based cryptosystems have incorporated more complicated protection mechanisms into existing encryption algorithms to defend against a brute-force attack, increasing the time required for every password guess in a brute-force attack. The MD5 crypt is one such example [11]. The MD5 hash can be brute-forced in a reasonable amount of time. However, applying the MD5 hash a thousand times increases the times required to crack the hash by the same factor. To overcome this protection method, password crackers have to design a good guessing mechanism that can guess effectively and obtain results in a limited amount of time.

Castelluccia [12] and Narayanan [13] proposed a password-cracking technique based on a Markov model, in which password guesses are based on the contextual frequency of characters. Weir et al. [14,15] presented a novel password-cracking technique that used the text structure from training data while applying mangling rules to the text itself. The authors found their technique to be more effective than the John the Ripper tool [16]. In a separate study, Zhang et al. [17] found Weir's algorithm to be the most effective among the techniques they used. In 2011, Kelley et al. [18] applied Weir's algorithm and a variation of the Markov model to generate simulated passwords and evaluate their strength.

However, Weir's algorithm inherently depends on common dictionaries. Weir et al. used three different dictionaries to generate passwords. Their method will not be effective if users have used words that are not in these dictionaries as passwords. For example, if the password is '!qazxsw@', based on the meaningless sequence of characters 'qazxsw', then Weir's algorithm will not be able to crack the password. Although 'qazxsw' is not a meaningful word, it is a keyboard pattern. This paper applies a variation of Weir's algorithm to generate more passwords more efficiently. We analyzed the patterns found in disclosed passwords, and developed the TDT model to generate passwords. These generated passwords were then used to crack UNIX access passwords.

3. **Pattern Analysis from Disclosed Passwords.**

User's password usually comes from printable characters. These characters are used in forming the passwords. We define the elements and password patterns obtained from the disclosed passwords.

3.1. **Basic elements and patterns of disclosed passwords.**

The basic elements of user's disclosed passwords are the printable characters. These characters available on a keyboard can be categorized into four different types, i.e., numeric (N), lowercase (L), uppercase (U) and other (O). Table 1 shows all the 94 printable characters and their type.

Table 1. Basic elements and types of users' passwords

Type	Character Number	Basic Elements
Numeric (N)	10	0123456789
Lowercase (L)	26	abcdefghijklmnopqrstuvwsyz
Uppercase (U)	26	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Other (O)	32	~ `!@#$% ^ &*()_−+=[]\{}|;':',./<>?

In order to describe and analyze the disclosed passwords comprising of N, L, U and O strings, a terminology is defined as follows.

Definition 3.1. *(N^+, N_n) A Number string (denoted N^+) is a continuous sequence of characters of N type. A Number string of length n (denoted N_n) is a continuous sequence of characters of N type of length n.*

Definition 3.2. *(Element) An element of type N_n is an instance of Number string N_n found in a disclosed password.*

For example, from an instance of disclosed passwords like '12348888', we obtain an element '12348888' of type N_8, which also belong to N^+. By Definition 3.1, N^+ represents either one of $N_1, N_2, N_3, \ldots, N_i$ and denotes a super class among $N_1, N_2, N_3, \ldots, N_i$.

Lemma 3.1. *The same definition also applies to Lowercase string (L^+, L_n), Uppercase string (U^+, U_n) and other string (O^+, O_n). An Alpha string (A^+, A_n) is a continuous sequence of characters of L or U types.*

Users create their passwords as patterns based on a combination among the strings of character types, defined as follows:

Definition 3.3. *(Password pattern and pattern class) The password pattern of a password is the combination of strings of character type (N, L, U and O) in a password. It is represented in terms of the combinations strings of N_n, L_n, U_n and O_n, where the subscript n represents the length of the corresponding type. The pattern class represents a set of password patterns and is a combination of strings of N^+, L^+, U^+ and O^+.*

For example, the password pattern of the password 'Password10!!' is '$U_1L_7N_2O_2$' since the password consists of four elements, i.e., 'P', 'assword', '10', and '!!'. These elements belong to patterns U_1, L_7, N_2, O_2, respectively. Furthermore, the password pattern '$U_1L_7N_2O_2$' is an instance of pattern class '$U^+L^+N^+O^+$'.

Some examples are presented in Table 2. We can see that most passwords are based on Alpha strings; they consist of Lowercase or Uppercase strings. Number strings or Other strings of variable length are inserted before or appended after the Alpha strings. Some Alpha strings are dictionary words, such as 'password', 'computer', 'tiger', and 'count'. Other passwords are based on the keyboard layout, e.g., 'qwerty', 'qazwsx'. In Section 3.2, we develop a password analysis platform to analyze users' disclosed passwords in detail.

Table 2. Some instances from users' disclosed passwords

Instance	Pattern Class	Password Pattern	Elements
password1	L^+N^+	L_8N_1	password, 1
computer88	L^+N^+	L_8N_2	computer, 88
13tmein	N^+L^+	N_2L_5	13, tmein
jester6#	$L^+N^+O^+$	$L_6N_1O_1$	jester, 6, #
Count!!!!	$U^+L^+O^+$	$U_1L_4O_4$	C, ount, !!!!
qwerty2009	L^+N^+	L_6N_4	qwerty, 2009
AAaa1211&&	$U^+L^+N^+O^+$	$U_2L_2N_4O_2$	AA, aa, 1211, &&
@merican	O^+L^+	O_1L_7	@, merican
95celica	N^+L^+	N_2L_6	95, celica
8qazwsx9	$N^+L^+N^+$	$N_1L_6N_1$	8, qazwsx, 9

3.2. **Statistics of password patterns.** In October 2008, a phishing attack was launched to obtain the email addresses and passwords of *Rockyou* users [19]. The phishing attack received much publicity, and the 14,344,389 passwords obtained were made available for download. However, the downloaded password corpus did not contain any information about associated *Rockyou* usernames. In the paper, we use these passwords as disclosed real password data for further analysis.

To analyze the password corpus more systematically, we developed a password analysis platform using Flex [5,6] with Cygwin [7]. The Cygwin package provides a complete UNIX environment from within a Windows PC. After installing Cygwin, Flex will be able to complete all flex assignments on a PC. Flex is normally used to partition a stream of characters into tokens. It takes a specification that associates regular expressions with actions as input. By using this password analysis platform with regular expressions, we analyze the passwords from *Rockyou* as described next.

Pattern Class, Password Pattern, and Element

According to our statistics, there are 141,278 pattern classes in *Rockyou*. However, 160 of these pattern classes are composed of continuous sequences of one to four character types, which cover 97.51% of these passwords in *RockYou*. For example, the password class of 'abc123' is 'L^+N^+', consisting of two character types. The reason is that overly complex passwords are difficult for users to remember. Therefore, we focus on calculating possible combinations of continuous sequences of one to four character types.

With our password analysis platform, we represent these 160 combinations by regular expressions, and use Flex to obtain statistics. Table 3 lists the top fifteen pattern classes in *RockYou* disclosed passwords.

TABLE 3. Top fifteen pattern classes from *Rockyou*

Pattern Class	Occurrences	Percentage of Total
L^+N^+	4,720,184	32.91%
L^+	3,726,129	25.98%
N^+	2,346,744	16.36%
N^+L^+	499,167	3.48%
$L^+N^+L^+$	388,157	2.71%
U^+N^+	325,941	2.27%
$U^+L^+N^+$	236,331	1.65%
U^+	229,875	1.6%
$L^+O^+L^+$	172,279	1.2%
$L^+O^+N^+$	144,129	1.0%
$L^+N^+L^+N^+$	123,344	0.86%
L^+O^+	122,560	0.85%
$N^+L^+N^+$	118,877	0.83%
U^+L^+	98,515	0.69%
$L^+N^+O^+$	65,164	0.45%

From Table 3, the top three pattern classes account for 75.25% of all passwords. This indicates that most users use a combination of Lowercase and Number strings, or just Lowercase strings or Number strings as their passwords. The top ranked 'L^+N^+' pattern class comprises 32.91% of the total. The statistical information regarding password class gives us a more detailed look into how users incorporate the four character types into passwords. For the 'L^+N^+' pattern class, we further calculate the password patterns according to various lengths of the Lowercase and Number strings. The result is shown

in Table 4. The most popular password pattern is 'L_6N_2', consisting of Lowercase string of six characters followed by Number string of two characters. The probability assigned to each password pattern is

$$\text{Probability}(x) = \frac{x_{\text{num}}}{K_{\text{tot}}}$$

where x_{num} is the number of occurrences of that particular password pattern x, and K_{tot} is the total number of 'L^+N^+' pattern class. For instance, the probability of password pattern 'L_6N_2' is $420318/4720184 = 8.91\%$. The same statistical information is recorded for each of the pattern class.

After obtaining the statistics of password patterns for each of the pattern class, we can sort the password patterns according to their probabilities. Next, we parse the passwords according to the previously defined four strings (N, L, U, O) and then count their probabilities respectively.

TABLE 4. Top ten password patterns in 'L^+N^+' pattern class from *Rockyou*

Password Patterns	Occurrences	Percentage of 'L^+N^+'
L_6N_2	420,318	8.91%
L_5N_2	292,306	6.19%
L_7N_2	273,624	5.80%
L_4N_4	235,360	4.99%
L_4N_2	215,074	4.56%
L_8N_2	213,109	4.51%
L_6N_1	193,097	4.10%
L_7N_1	189,847	4.02%
L_5N_4	173,559	3.68%
L_6N_4	160,592	3.40%

Number strings in Passwords

The selection of Number strings in passwords often represents something of significance, such as birthday, telephone number or special date. The patterns of Number strings are been classified dependent upon their lengths in passwords. Table 5 describes the statistics of length between one and ten in Number strings. The pattern of length two is the most frequent as its occurrence is 24.38%. 65.35% of these patterns have lengths between one and four for Number strings. Each pattern of Number strings is analyzed to determine which elements are most frequently seen in passwords. Table 6 shows the top ten elements of length two of pattern 'N_2'. The most frequent element is '12', with an occurrence of 4.81%. Similar statistics are recorded for each of pattern in Number strings.

Other strings in Passwords

Other strings are also important in password generation. The same process that is detailed above for Number strings is repeated for Other strings. The patterns of length between one and ten in Other strings occur a total of 301,096 times. Table 7 displays the frequencies of each pattern of length between one and ten for Other Strings. It is clear that pattern of length one occurs more frequently than any others do. Table 8 describes the most frequent element of length one occurring in pattern 'O_1'. The same statistics are collected for the other patterns of Other strings and recorded.

Alpha Strings in Passwords

Alpha strings are the main components in creating a password, which are comprised of Lowercase strings and Uppercase strings. In order to enhance the strength of passwords, usually the passwords formed by familiar Alpha strings are combined with Number strings

TABLE 5. Patterns of length between one and ten in Number strings

Patterns of Number Strings	Occurrences	Percentage of Total
N_2	2,131,337	24.38%
N_4	1,330,610	15.22%
N_1	1,321,553	15.12%
N_3	928,760	10.63%
N_6	776,436	8.88%
N_7	608,993	6.97%
N_8	523,130	5.98%
N_{10}	503,930	5.77%
N_9	343,396	3.93%
N_5	272,088	3.11%

TABLE 6. Elements of length two of pattern 'N_2' in Number strings

Elements of Length Two	Occurrences	Percentage of 'N_2'
12	102,590	4.81%
13	76,775	3.60%
11	65,201	3.06%
22	58,058	2.72%
23	57,825	2.71%
07	55,693	2.61%
21	55,192	2.59%
01	53,608	2.52%
10	53,510	2.51%
14	53,300	2.50%

TABLE 7. Patterns of length between one and ten in Other strings

Patterns of Other Strings	Occurrences	Percentage of Total
O_1	238,652	79.26%
O_2	38,780	12.88%
O_3	16,184	5.37%
O_4	3,544	1.18%
O_6	1,260	0.42%
O_5	1,177	0.38%
O_7	623	0.21%
O_8	452	0.15%
O_9	226	0.08%
O_{10}	198	0.07%

and Other strings. For example, password 'computer123', consists of the Alpha string 'computer' and Number string '123'. In this paper, we take Alpha strings as the basis and associate them with Number strings and Other strings to generate new passwords. Therefore, we parse out the Alpha strings from *Rockyou* and use the set as the Dictionary set, which is illustrated in Section 4. The total number of unique Alpha strings from *Rockyou* is 5,521,967.

According to our observation for these parsed Alpha stings, many users still select familiar dictionary words. In order to count the number of dictionary words in Alpha

TABLE 8. Elements of length one in pattern 'O_1'

Elements of Length One	Occurrences	Percentage of 'O_1'
!	64,602	27.07%
.	42,446	17.79%
*	33,950	14.23%
@	13,995	5.86%
$	10,941	4.58%
_	8,393	3.52%
-	6,922	2.90%
?	6,447	2.70%
(	6,085	2.55%
)	4,818	2.02%

TABLE 9. Statistics of dictionary words in alpha strings in *Rockyou*

Alpha strings	Numbers of Occurrences	Percentage of Total
Dictionary words	1,254,021	8.74%

strings, we collected 869,228 dictionary words from Dic-0294 [20] for reference. To enhance the efficiency of comparison, the MD5 hashing algorithm is used to process the large number of Alpha strings. Tests show this provides a ten-time speedup. The statistics are given in Table 9. It proved that many users still select weak dictionary words to form their passwords.

Keyboard Strings in Passwords

In this paper, we also analyze the password patterns generated by the keyboard positions. The Keyboard strings refer to passwords formed by relative keyboard positions and associated finger movements. Its examples include 'keyboard neighbor' and 'keyboard parallel n', which are parallel patterns of length n. In this paper, such passwords are referred to as Keyboard strings, and represented by the symbol K. For example, the Keyboard string '1qa2ws' is formed by a keyboard parallel 3 pattern on the keyboard. Using Flex, we overcome the limitation on recursive expressions, and use regular expressions to analyze keyboard neighbor and parallel relationships. The statistics are given in Table 10. Notice that although the percentage of keyboard strings is not high, with increasing awareness and emphasis on password security, more and more users choose this structure for their passwords.

TABLE 10. Statistics of keyboard strings in *Rockyou*

Keyboard String	Example	Numbers of Occurrences	Percentage of Total
Keyboard Neighbor	asdfvcxz	20,739	0.15%
Keyboard Parallel 3	1qa2ws	1898	0.013%
Keyboard Parallel 4	0okm9ijn	780	0.0054%
Keyboard Parallel 5	12345qwert	231	0.0016%

Using our password analysis platform, we analyzed the passwords from *Rockyou* and found 176 common and popular password patterns through Flex. These patterns cover 96.3% of the passwords in *RockYou* corpus. With this analysis platform, the patterns of users' disclosed passwords can be analyzed quickly. Moreover, this platform is also easy to maintain and extend as password patterns can be added and removed. For instance, if in the future we find more than four character types change used in passwords, e.g.,

'!!!Password88##', containing '$O_3U_1L_7N_2O_2$', i.e., a continuous sequence of five character types, we just need to add the regular expression that describes the new pattern to Flex before performing analysis. In Section 4, we use these statistics of the password patterns as a basis to generate efficient passwords for reducing the number of guesses and increasing the hit rate.

4. **Strategy for Generating Passwords.** From statistics of disclosed passwords for *RockYou* described in Section 3, we summarize some common patterns for selecting passwords.

1) Many passwords are composed of Lowercase and Number strings, i.e., 'L^+N^+', the Lowercase strings are likely appended by Number strings.
2) Number or Other strings are inserted before or appended after Alpha strings, e.g., 'jessica!!' belongs to 'L_8O_2' pattern, the Alpha string 'jessica' is appended with the Other string '!!'.
3) Alpha strings include Keyboard strings, e.g., 'qwerty123' belongs to 'K_6N_3' pattern, containing the Keyboard string 'qwerty'.
4) Alpha strings include dictionary words, e.g., 'football1234' belongs to 'L_8N_4' pattern, containing the dictionary word 'football'.

Instead of trying to create rules that mimic these common password patterns, we assign probabilities to these patterns, and then use those probabilities directly to generate very fine-grained rules, and thus generate passwords for cracking.

4.1. **TDT model.** Based on the analysis of the disclosed passwords in Section 3, we create the TDT model consisting of the Training set, Dictionary set and Testing set, to generate the passwords. Three sets in the model are defined as follows.

Definition 4.1. *(Training set, Dictionary set and Testing set) The Training set specifies users' disclosed passwords. The Dictionary set specifies dictionary words, Alpha strings or Keyboard strings. The Testing set specifies the cracking targets, which are plaintext passwords. If a password is used in the Training set, we ensure that it is not included in the Testing set.*

The TDT model uses the Training set to create password rules, and use these rules along with the Dictionary set to generate passwords. Then it uses the generated passwords to crack passwords in the Testing set. Note that a password is considered 'cracked' if our method generates a guess that matches a password in the Testing set. The framework of the TDT model is shown in Figure 1 and is described below.

1) Disclosed passwords collection: In our method, user disclosed passwords are used as the Training set, collected from various sources (e.g., specific websites) or from broken passwords of target. For this study, we simply use the *Rockyou* password corpus as the Training set.
2) Establishing UDP rules: When sufficient passwords have been collected, the Training set is used to create user disclosed passwords rules (shortened as UDP rules).
3) Passwords generation: The passwords could be generated using the UDP-rules along with the Dictionary set. The Dictionary set consists of the commonly used dictionary words or Alpha and Keyboard strings parsed from the Training set.
4) Password cracking and feedback: The generated passwords are used to crack passwords from the Testing set. Once passwords have been cracked, they can be fed back to the Training set. When the Training set is updated, we repeat steps 2-4, to update UDP rules and generate new passwords. Through this repeated learning and training

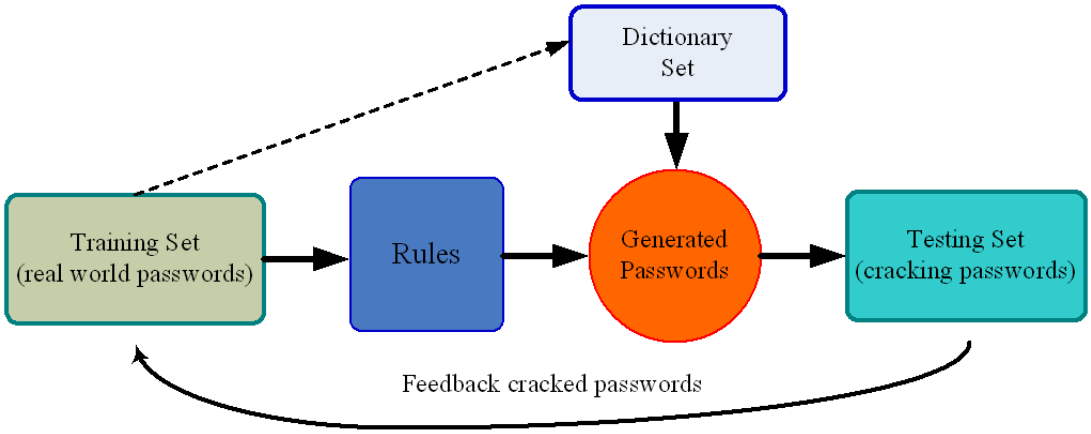

FIGURE 1. The framework of TDT model

process, we can optimize the accuracy of the UDP rules and enhance the guessing ability of the model.

4.2. **Establishing UDP rules.** Each password in the Training set is parsed. The parsed strings in a password that have Number strings and Other strings are called 'leaf nodes' and those having Lowercase, Uppercase, and Keyboard strings are called 'inner nodes'. Leaf nodes are categorized based on character types and their length. Their probabilities are calculated based on the number of times they occur in the Training set. The probability of each leaf node is calculated as follows.

$$\frac{number\ of\ occurrences\ of\ specific\ leaf\ node\ of\ length\ k}{n_k}$$

where n_k is the total number of strings of length k. For example, if the Number strings of length 3 are '123' and '456', and they occur in the Training set 100 and 60 times respectively, then the probability for string '123' is $100/160 = 0.625$ and that for string '456' is $60/160 = 0.375$. Inner nodes are used as dictionary words and added to the Dictionary set. At this point, dictionary words in the Dictionary set do not have probabilities associated with them, so we set the probability for inner nodes is 1.0.

Moreover, we use the symbols N_n, L_n, U_n, O_n, K_n defined in Section 3 to represent each password pattern, where N stands for Number, L for Lowercase, U for Uppercase, O for Other, and K for Keyboard, and the subscript n represents the length of the corresponding symbols. For example, the password pattern of 'abc123' is 'L_3N_3'. Based on this rule, passwords with the same patterns will be merged, and the probability of each pattern can be calculated. Thus, we can calculate the probability of each password pattern and find which among them are used more frequently.

We describe this process through a simple example. Assume that there are eight passwords in the Training set as shown in Figure 2. For leaf nodes, we categorize them by length, and then calculate their probabilities. Among strings of length = 1, the Other string '@' appears twice, and '!' appears once. Therefore, the probability of '@' is $2/3 = 0.67$, while that of '!' is $1/3 = 0.33$. Using the same method, we can find the probabilities of Number strings of length three and four, as shown in Figure 2(a). Inner nodes are used as dictionary words, as explained in Section 4.3. In addition, after converting the eight passwords into their patterns, we merge these patterns and obtain five distinct patterns, as shown in Figure 2(b). For example, the three passwords 'abc123',

'cat789' and 'man666' all have the 'L_3N_3' pattern, so its probability is $3/8 = 0.375$. Note that the higher the probability of a pattern, the more frequently it is used by users.

4.3. **Generating passwords.** Using the UDP rules created in Section 4.2, we generate passwords with the help of a Dictionary set. The Dictionary set mainly consists of commonly used dictionary words, or the inner nodes parsed from the Training set. By constructing the Dictionary set, we can gain a larger coverage of the target set of passwords. The core idea of this paper is to generate highly efficient passwords by using the Dictionary set as the basis and combining it with rules generated from the Training set. An example Dictionary set is shown in Figure 3. In addition to the eight inner nodes parsed from the Training set in Section 4.2, we added the three words 'princess', 'nicole' and 'you' to the Dictionary set.

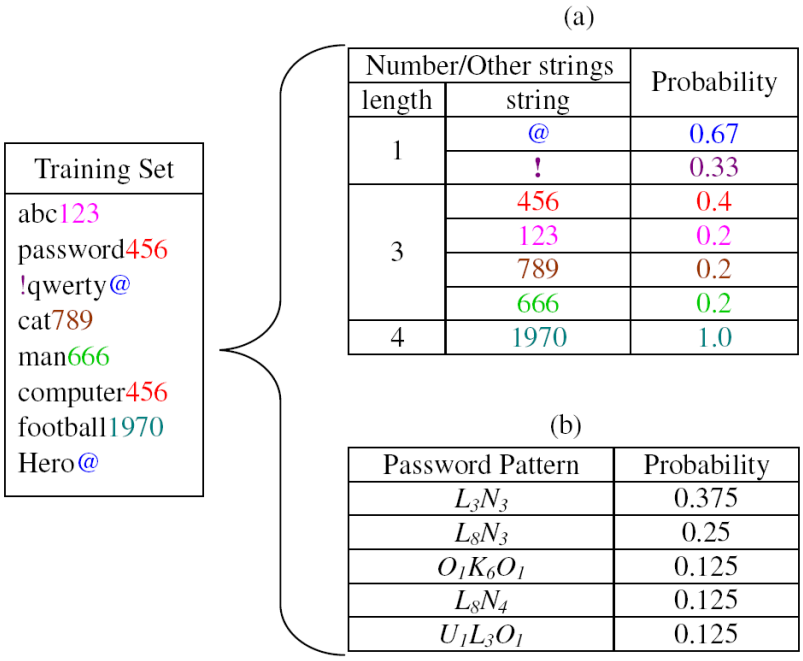

FIGURE 2. Examples of generated UDP rules using a training set

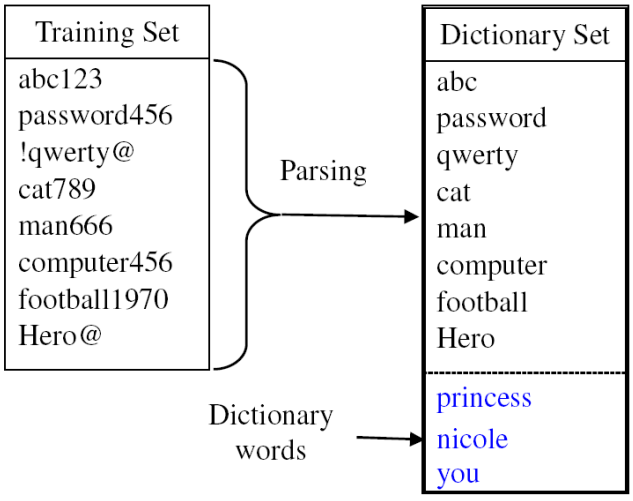

FIGURE 3. The dictionary set

Next, we use the Dictionary set and the UDP rules to generate passwords and calculate their probabilities. The formal mathematical definition is as follows.

Theorem 4.1. *Assume that $P(\beta_x)$ be the probability of the pattern of a password x, which $\beta_x = t_1 t_2 t_3 \ldots t_n$ is the pattern of a password x, consisting of $t_1, t_2, t_3, \ldots, t_n$ inner or leaf nodes. Let $v_{\ell_i}(t_i)$ be the probability of the i-th inner or leaf nodes of length ℓ in the Training set. Therefore, the probability of generated password x is given by*

$$P(x) = P(\beta_x) \cdot \prod_{i=1}^{n} v_{\ell_i}(t_i),\ t_i \in \{\textit{inner nodes, leaf nodes}\}$$

We use the generated password 'abc456' as an example. Since the string 'abc' in the Dictionary set matches 'L_3' in the password pattern 'L_3N_3', it must be combined with Number strings in the form of 'N_3' to generate a password. Therefore, the probability of generated password 'abc456' is

$$\begin{aligned} P(abc456) &= P(\beta_{abc456} = t_1 t_2 = L_3 N_3) \cdot v_{\ell_1}(L_3) \cdot v_{\ell_2}(N_3) \\ &= 0.375 \cdot 1.0 \cdot 0.4 \\ &= 0.15 \end{aligned}$$

For the same method, we calculate the probability of each password based on the respective probabilities of Number strings '123' '789' and '666', as shown in Table 11.

TABLE 11. Probability of generated passwords using dictionary set and UDP rules

Dictionary set	Pattern	Number String	Probability of Generated Passwords	Generated Passwords
	Probability	Probability		
abc	L_3N_3	456	$0.375 * 0.4 = 0.15$	abc456
	0.375	0.4		
abc	L_3N_3	123	$0.375 * 0.2 = 0.075$	abc123
	0.375	0.2		
abc	L_3N_3	789	$0.375 * 0.2 = 0.075$	abc789
	0.375	0.2		
abc	L_3N_3	666	$0.375 * 0.2 = 0.075$	abc666
	0.375	0.2		

Since the probability of each generated password can be calculated, we sort the probabilities of generated passwords in decreasing order and apply a threshold θ. Therefore, the generated passwords from the TDT model are

$$D_{v,\theta} = \left\{ X : P(\beta) \cdot \prod_{i=1}^{n} v_{l_i}(t_i) \geq \theta,\ t_i \in \{\textit{inner nodes, leaf nodes}\} \right\}$$

Table 12 shows the top twenty passwords along with their probabilities, as generated using the UDP rules in Section 4.2 along with our Dictionary set. The threshold θ is set as 0.075. One of the advantages of our method is the ability to sort the generated passwords according to their probabilities. In this way, based on user requirements, we can set a probability threshold and filter out passwords having low probabilities, and thus reduce the guessing space. When a password is successfully cracked by our method, we can move the password back to the Training set, and rebuild the UDP rules. Through this repeat training mechanism, the generated passwords become better adapted to the targets, increasing the hit rate.

TABLE 12. Top 20 generated passwords using UDP rules and dictionary set

Rank	Generated passwords	Probability	Rank	Generated passwords	Probability
1	abc456	$0.375 * 0.4 = 0.15$	11	password456	$0.25 * 0.4 = 0.1$
2	cat456	$0.375 * 0.4 = 0.15$	12	princess456	$0.25 * 0.4 = 0.1$
3	man456	$0.375 * 0.4 = 0.15$	13	Hero@	$0.125 * 0.67 = 0.084$
4	you456	$0.375 * 0.4 = 0.15$	14	abc666	$0.375 * 0.2 = 0.075$
5	computer1970	$0.125 * 1.0 = 0.125$	15	abc789	$0.375 * 0.2 = 0.075$
6	football1970	$0.125 * 1.0 = 0.125$	16	abc123	$0.37 * 0.2 = 0.075$
7	password1970	$0.125 * 1.0 = 0.125$	17	cat666	$0.375 * 0.2 = 0.075$
8	princess1970	$0.125 * 1.0 = 0.125$	18	cat789	$0.375 * 0.2 = 0.075$
9	computer456	$0.25 * 0.4 = 0.1$	19	cat123	$0.375 * 0.2 = 0.075$
10	football456	$0.25 * 0.4 = 0.1$	20	man666	$0.375 * 0.2 = 0.075$

5. **Experiments and Results.**

5.1. **Password generation and cracking.** For a better computational complexity, we divided the 14,344,389 leaked *Rockyou* passwords randomly into 28 subsets using the GNU *shuf* tool. Each such subset consisted of 500,000 passwords. For convenience, we selected the first and last subset as the Training set and Testing set respectively, called Rockyou1 and Rockyou28. The Dictionary set used is Dic-0294, which is a commonly used password cracking dictionary [20]. This dictionary contains 869,228 unique words. To evaluate performance, we selected Rockyou28, Myspace, and Phpbb plaintext passwords as the Testing set. The overall structure of TDT model is shown as Figure 4. The Myspace and Phpbb plaintext passwords are also created by attackers who had performed a phishing attack against users but failed to secure their collection server. Researchers proceeded to log into the malicious server and gathered the passwords from the hacker before the hacker could shut it down. The complete Myspace and Phpbb contained 36,764 and 184,380 plaintext passwords respectively. A summary is given in Table 13, and experiment results are presented in Figure 5.

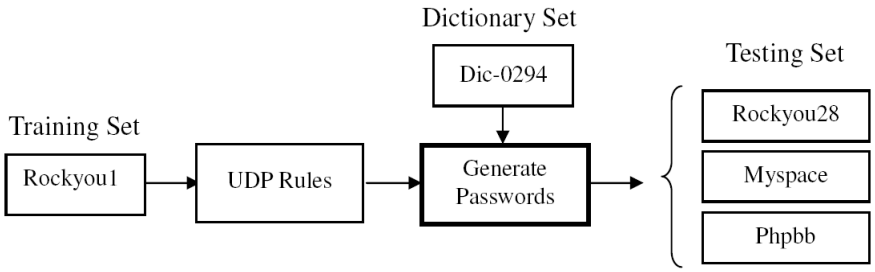

FIGURE 4. Overall structure of TDT model

Figure 5 shows that our method is more efficient for password cracking. When the number of generated passwords is between 50 and 450 million, the hit rate for Myspace is 27-33%, for Phpbb is 26-31%, and for Rockyou28 is 20-28%. The results show that the hit rate for Rockyou28 is lower than that for Myspace and Phpbb. In other words, password strength in the Rockyou28 set is higher than that in Myspace and Phpbb. Password strength differs with users' cultural backgrounds, age, password creation requirements, importance of the password protected material, etc.

TABLE 13. The training, dictionary and testing sets

Set	Name	Number
Training set	Rockyou1	500,000
Dictionary set	Dic-0294	869,228
Testing set	Rockyou28	500,000
	Myspace	36,764
	Phpbb	184,389

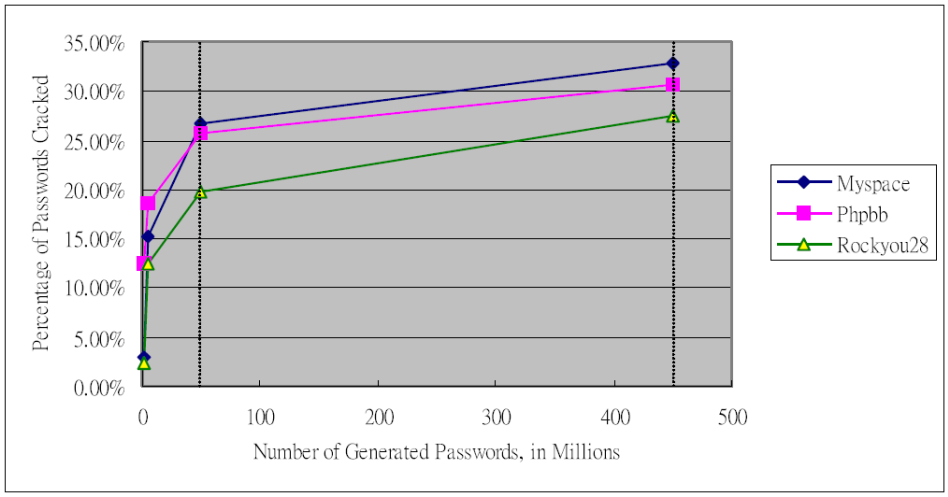

FIGURE 5. Percentage of passwords cracked

In Figure 5, we only use common used dictionary words (Dic-0294) as the basis for the Dictionary set. This procedure is suitable when we know nothing about the target passwords (i.e., the Testing set). However, we may have access to some passwords of specific targets from various channels. Having access to the partial list of target passwords, these passwords may be used to generate rules, and to parse the inner nodes from these passwords, combining them with the Dictionary set. In the above experiment, since Rockyou1 and Rockyou28 originate from the same source (Rockyou), we can assume they both have similar underlying password creation habits. Therefore, we consider Rockyou1 as the cracked passwords of Rockyou (i.e., the Training set), and consider Rockyou28 as the uncracked passwords of Rockyou (i.e., the Testing set). After parsing the inner nodes from Rockyou1 and combining them with the Dictionary set, the generated passwords should help increase the hit rate for Rockyou28.

To verify this prediction, we parsed out 284,903 inner nodes that included Lowercase, Uppercase, and Keyboard strings from Rockyou1, and call it Rockyou1_dic. These strings are added to Dic-0294 to form a new Dictionary set, which contained 1,154,131 dictionary words. The structure is shown in Figure 6 and the number of sets is listed in Table 14. We repeated the same experiments again and the experimental results are shown in Figure 7. When the number of generated passwords is between 50 and 450 million, the hit rate compared with Figure 5, increases from 27-33% to 30-36% for Myspace, from 26-31% to 29-34% for Phpbb, and from 20-28% to 32-41% for Rockyou28. It is apparent that the performance on Rockyou28 shows the most improvement. This experiment shows that our UDP model can be customized for a specific target. When we add the inner nodes of Rockyou1 to the Dictionary set, the generated passwords have a higher chance of

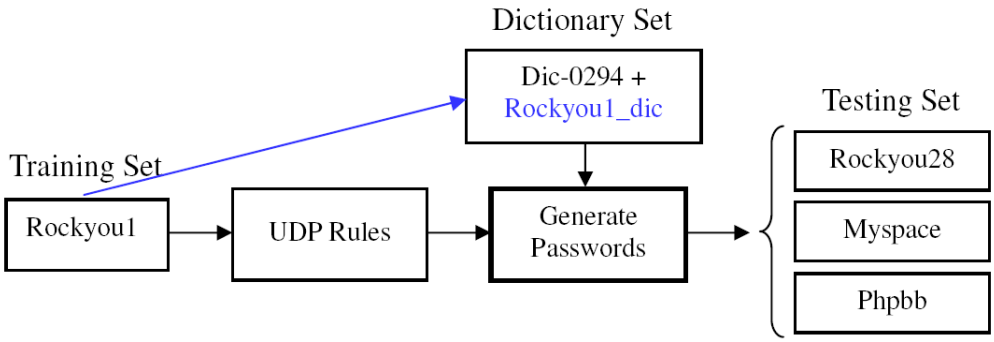

FIGURE 6. Overall structure of TDT model

TABLE 14. The training, dictionary and testing sets

Set	Name	Number	
Training set	Rockyou1	500,000	
Dictionary set	Dic-0294	869,228	1,154,131
	Rockyou1_dic	284,903	
Testing set	Rockyou28	500,000	
	Myspace	36,764	
	Phpbb	184,389	

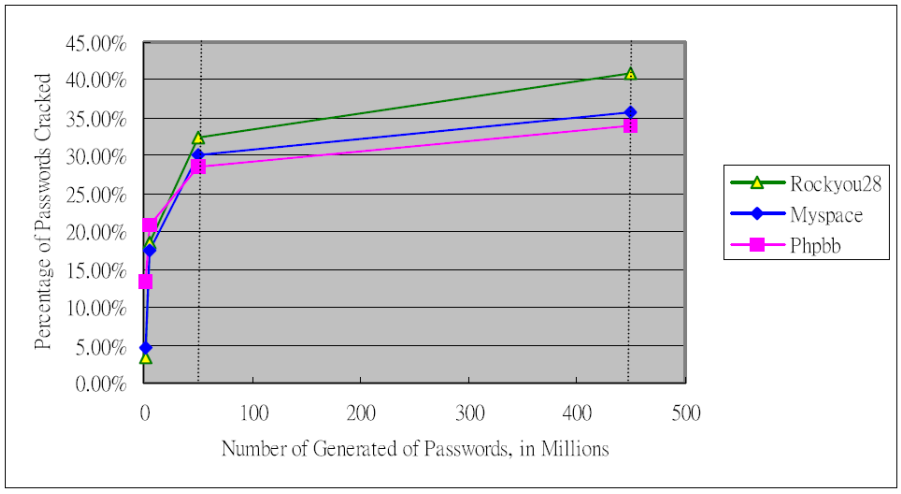

FIGURE 7. Percentage of passwords cracked. The Dictionary set consists of Dic-0294 and Rockyou1_dic.

matching passwords in Rockyou28. This implies that if we know some password patterns of a specific target, the hit rate of password cracking can be improved.

To summarize the experimental results from Section 5.1, our method is suitable for two situations, described as follows:

- "Generalization" condition: where nothing is known about the target passwords. The Training set includes disclosed passwords (such as from Rockyou1) and the Dictionary set only includes commonly used dictionary words (e.g., Dic-0294). Figure 5 shows that using this information to crack passwords of Myspace, Phpbb and Rockyou28, a hit rate of 20%-33% can be achieved for 50 and 450 million generated passwords.

- "Customization" condition: where some information about the target passwords is known. The Training set includes the cracked target passwords (such as Rockyou1) and the Dictionary set includes the inner nodes parsed from these previously cracked passwords, as well as commonly used dictionary words (e.g., Dic-0294). Figure 7 shows that this method increases the hit rate of Rock28 from 20-28% to 32-41%.

5.2. **Experimental observations.** Figure 8 shows a comparison of our method with the John-the-Ripper cracker tool [16] and Brute-force attack. The experiment used Rockyou28 as the Testing set. The result shows that the same number of generated passwords, our method cracked significantly more passwords than the John-the-Ripper and Brute-force attack. For example, for 300 million generated passwords, our method cracked 143% of the passwords cracked using John-the-Ripper, and 250% of those cracked using Brute-force attack.

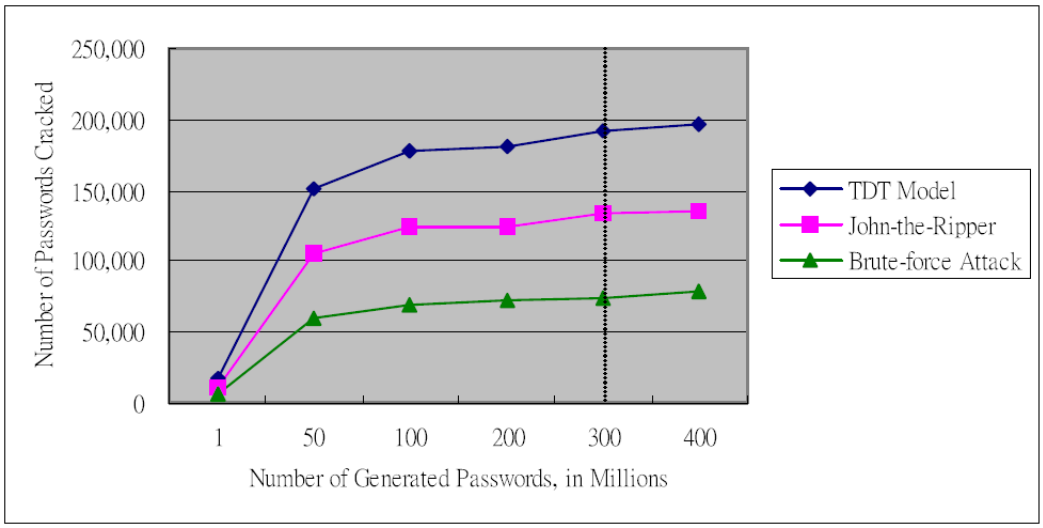

FIGURE 8. Comparison with john-the-ripper and brute-force attack

Based on the above experimental results, the advantages of the proposed method are as follows:

1) The proposed method has self-training and learning capabilities. When more passwords of the target are known, more accurate passwords can be generated using the training and learning process.
2) Experiments showed that many users still use Alpha strings as the basis of their passwords, which are combined with variable-length Number or Other strings at the beginning or end of the Alpha strings.
3) In the past, people often used dictionary words for Alpha strings. However, Alpha strings among the disclosed passwords are increasingly keyboard words or other meaningless words. The proposed method is good at guessing such words as we add these words to the Dictionary set.
4) Based on the same number of generated passwords, our method can crack more passwords than the John-the-Ripper and Brute-force attack, as shown in Figure 8.
5) This method is designed for the English language at present. However, it can also be extended to other languages such as Chinese and Spanish.

6. **Application.** This section describes a hybrid password cracking system to crack access passwords collected from UNIX. Within this system, we introduce a new stage called TDT-model attack, which uses the passwords generated by TDT model.

6.1. **Effectiveness.** The design goal of the hybrid password cracking system is to crack encrypted target passwords by incrementally increasing the size of the password cracking space. Therefore, the system consists of three attack stages: the Dictionary attack followed by the TDT-model attack, and the Brute-force attack; this system is called the DTB password cracking system, as shown in Figure 9. The cracking strategy is to crack encrypted passwords using the Dictionary attack and the TDT-model attack first in finite time, and to then run the Brute-force attack when some encrypted passwords have not been cracked. The system can be used in DTB mode where all stages are enabled or in DB mode where only the TDT-model attack stage is disabled.

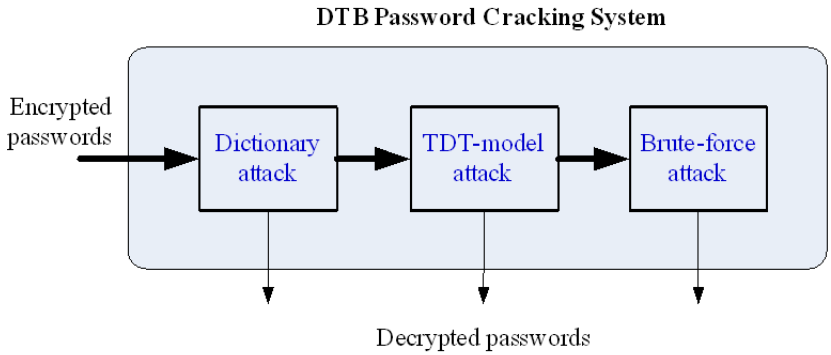

FIGURE 9. The system architecture of DTB password cracking system

UNIX Access Password Attack

We collected 382 encrypted access passwords for the UNIX system using the John-the-Ripper cracker tool [16]. These passwords are of 13-byte values generated by the DES cryptosystem. Then, we proceeded to crack the passwords in DTB and DB modes. The results showed that, in the same 5-hour experiments, 226 passwords and 155 passwords are cracked in DTB mode and DB mode, respectively. After the Dictionary attack, in the same period of time, the number of passwords cracked with the TDT-model increases by 71 and by up to 273% [(75+37)/41]. The details are shown in Table 15, where the time taken to crack the passwords is given in brackets.

TABLE 15. Number of UNIX passwords cracked

DTB Password Cracking System	DTB mode			DB mode	
	Dictionary attack	TDT-model attack	Brute-force attack	Dictionary attack	Brute-force attack
Number of Passwords Cracked	114 (12min)	75 (13min)	37 (4hr 35min)	114 (12min)	41 (4hr 48min)
	226			155	

6.2. **Comparison.** Among the dictionary-based password cracking methods, Weir et al. [14,15] focused primarily on creating a probabilistic context-free grammar to generate word-mangling rules based on previously disclosed passwords. In 2011, Kelley et al. [18] proposed an efficient technique for evaluating password strength that is also based on the Weir's algorithm. However, Weir's algorithm will not be effective if users have not used passwords in their inherently common dictionaries. In the TDT model, we utilized Number, Lowercase, Uppercase, Other and Keyboard strings to generate rules. Moreover,

the TDT model may parse Lowercase, Uppercase, and Keyboard strings from the target's known passwords and combine these strings with the set of dictionary words, effectively generating "customized" passwords for the target. In addition, cracked passwords in the TDT model can be fed back into the Training set, and training rules can be reapplied. Table 16 showed a comparison of the methods of Weir, Kelley, and the TDT model.

TABLE 16. Comparisons of dictionary-based password cracking methods

Method	Weir	Kelley	TDT model
Feature	Password structure formally modeled using a context-free grammar.	Measured password strength by simulating Weir's algorithms.	Developed the probability-based password cracking algorithm, a superset of Weir's patterns; verified in real cracking system.
Execution	Offline.	Offline.	Offline/Online.
Flexibility	'General' passwords generated.	'Specific' passwords generated through different composition policies.	'Customized' passwords can also be generated.
Limitation	Password generation depend inherently on common dictionaries.	Password strength depends on the specific algorithm.	Password generation based on known password patterns.

Figure 10 compares the cracking performance of the Weir method and our TDT model. The Myspace list containing 36,764 passwords used in the experiment was divided equally into a Training set and a Testing set. Dic-0294 was used as the Dictionary set. The performance of the TDT Model is clearly better than the Weir method.

Passwords generated using the TDT model are based on users' disclosed passwords and dictionary words, and can be used widely for cracking password-based cryptosystems. To resist attacks, many current password-based cryptosystems not only strengthen their cryptographic security, but also force users to create 'strong' passwords, which must at least be a combination of two character types and have a minimum length of eight characters. The strong passwords fall within the password space of the TDT model, and therefore, the cracking time is reduced.

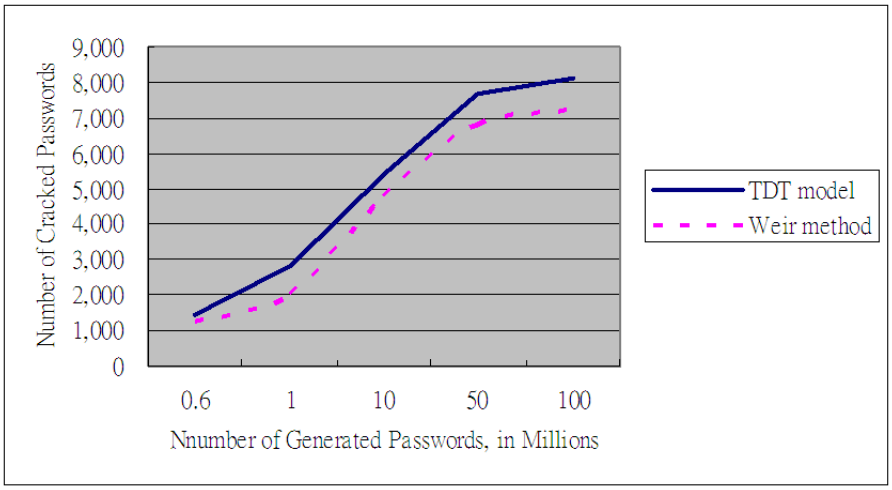

FIGURE 10. Comparison between the Weir method and TDT model

7. **Conclusion and Future Work.** We developed a password analysis platform to formally analyze disclosed passwords, and establish the TDT model for generating passwords. Based on the probabilistic patterns of the TDT model, the passwords generated are sorted in decreasing order of probability; this is known as the TDT-model attack. We also design a hybrid password cracking system consisting of the Dictionary attack, TDT-model attack and Brute-force attack to verify the effectiveness of the TDT model. Experimental results show that this hybrid model is effective for cracking UNIX access passwords and there is a performance improvement of up to 273% from using the TDT-model attack. In future work, we plan to focus on letter replacement, such as replacing 'a' in a dictionary word with '@' to further strengthen passwords by providing more flexibility and generality. The effectiveness of this strategy with other languages may also be studied.

Acknowledgement. The authors gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved this paper. The paper received a grant from the National Science Council, NSC 100-2219-E-002-032.

REFERENCES

[1] H. Gao, X. Liu, S. Wang, H. Liu and R. Dai, Design and analysis of a graphical password scheme, *International Conference of Innovative Computing, Information and Control*, Xi'an, China, pp.675-678, 2009.

[2] S. Delaune and F. Jacquemard, A theory of dictionary attacks and its complexity, *Proc. of the 17th IEEE Computer Security Foundations Workshop*, 2004.

[3] J. Yan, A. Blackwell, R. Anderson and A. Grant, Password memorability and security: Empirical result, *IEEE Security and Privacy Magazine*, vol.2, no.5, pp.25-31, 2004.

[4] R. V. Yampolskiy, Analyzing user password selection behavior for reduction of password space, *Proc. of the IEEE International Carnahan Conferences on Security Technology*, pp.109-115, 2006.

[5] Flex, *The Fast Lexical Analyzer*, http://flex.sourceforge.net/, 2007.

[6] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 1979.

[7] G. J. Noer, *Cygwin32: A Free Win32 Porting Layer for UNIX® Applications*, 1998.

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

[9] V. L. L. Thing and H. M. Ying, A novel time-memory tradeoff method for password recovery, 2009.

[10] O. Billet and H. Gilbert, Cryptanalysis of rainbow, *Security and Cryptography for Networks*, vol.4116, pp.336-347, 2006.

[11] *Security Issues with MD5*, http://en.wikipedia.org/wiki/MD5#Security.

[12] C. Castelluccia, M. Dürmuth and D. Perito, Adaptive password-strength meters from Markov models, *Proc. of the Network and Distributed System Security Symposium*, 2012.

[13] A. Narayanan and V. Shmatikov, Fast dictionary attacks on passwords using time-space tradeoff, *Proc. of the 12th ACM Conference on Computer and Communications Security*, 2005.

[14] M. Weir, S. Aggarwal, B. de Medeiros and B. Glodek, Password cracking using probabilistic context-free grammars, *Proc. of the 30th IEEE Symposium on Security and Privacy*, pp.391-405, 2009.

[15] C. M. Weir, *Using Probabilistic Techniques to Aid in Password Cracking Attacks*, Ph.D. Thesis, Florida State University, 2010.

[16] Openwall Project, *John the Ripper Password Cracker*, 2010.

[17] Y. Zhang, F. Monrose and M. K. Reiter, The security of modern password expiration: An algorithmic framework and empirical analysis, *Proc. of the 17th ACM Conference on Computer and Communications Security*, pp.176-186, 2010.

[18] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor and J. Lopez, Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms, *Tech. Rep. CMU-CyLab-11-008*, Carnegie Mellon University, 2011.

[19] *Rockyou Users*, http://www.skullsecurity.org/wiki/index.php/Passwords.

[20] *http://www.linux-pour-lesnuls.com/traduc/Dictionnaires/dic-0294/dic-0294.txt.*