A Novel Rainbow Table Sorting Method

Hwei-Ming Ying, Vrizlynn L. L. Thing

Cryptography & Security Department
Institute for Infocomm Research, Singapore
{hmying,vriz}@i2r.a-star.edu.sg

***Abstract*—As users become increasingly aware of the need to adopt strong password, it also brings challenges to digital forensics investigators due to the password protection of potential evidence data. In this paper, we discuss existing password recovery methods and propose a new password sorting method that aid in improving the performance of the recovery process. This improved method supports a quick binary search instead of the slower linear search as employed in the enhanced rainbow table. We show that this method will result in a 23% reduction in storage requirement, compared to the original rainbow tables, while maintaining the same success rate. It is also an improvement over the enhanced rainbow table as the time taken for the password lookup will be drastically reduced.**

Keywords - Digital forensics; password recovery; search optimization; time-memory tradeoff; cryptanalysis.

I. Introduction

In computer and information security, the use of passwords is essential for users to protect their data and to ensure a secured access to their systems/machines. However, in digital forensics, the use of password protection presents a challenge for investigators while conducting examinations. As mentioned in [1], compelling a suspect to surrender his password would force him to produce evidence that could be used to incriminate him, thereby violating his Fifth Amendment right against self-incrimination. Therefore, this presents a need for the authorities to have the capability to access a suspect's data without expecting his assistance. While there exist methods to decode hashes to reveal passwords used to protect potential evidence, lengthier passwords with larger characters sets have been encouraged to thwart password recovery. Awareness of the need to use stronger passwords and active adoption have rendered many existing password recovery tools inefficient or even ineffective.

The more common methods of password recovery techniques are guessing, dictionary, brute force and more recently, using rainbow tables. The guessing method is attempting to crack passwords by trying "easy-to-remember", common passwords or passwords based on a user's personal information (or a fuzzy index of words on the user's storage media). A statistical analysis of 28,000 passwords recently stolen from a popular U.S. website revealed that 16% of the users took a first name as a password and 14% relied on "easy-to-remember" keyboard combinations [2]. Therefore, the guessing method can be quite effective in some cases where users are willing to compromise security for the sake of convenience.

The dictionary attack method composes of loading a file of dictionary words into a password cracking tool to search for a match of their hash values with the stored one. Examples of password cracking tools include Cain and Abel [3], John the Ripper [4] and LCP [5].

In the brute force cryptanalysis attack, every possible combination of the password characters is attempted to perform a match comparison. It is an extremely time consuming process but the password will be recovered eventually if a long enough time is given. Cain and Abel, John the Ripper as well as LCP are able to conduct brute force attacks.

In [6-9], the authors studied on the recovery of passwords or encryption keys based on the collision of hashes in specific hashing algorithms. These methods are mainly used to research on the weakness of hashing algorithms. They are too high in complexity and time consuming to be used for performing password recovery during forensics investigations. The methods are also applicable to specific hashing algorithms only.

In [10], Hellman introduced a method which involves a trade-off between the computation time and storage space needed to recover the plaintext from its hash value. It can be applied to retrieve Windows login passwords encrypted into LM or NTLM hashes [11], as well as passwords in applications using these hashing algorithms. Passwords encrypted with hashing algorithms such as MD5 [12], SHA-2 [13] and RIPEMD-160 [14] are also susceptible to this recovery method. In addition, this method is applicable to many searching tasks including the knapsack and discrete logarithm problems.

In [15], Oechslin proposed a faster cryptanalytical time-memory trade-off method, which is an improvement over Hellman's method. Since then, this method has been widely used and implemented in many popular password recovery tools. The pre-computed tables that are generated in this method are known as the rainbow tables.

In [16], Narayanan and Shmatikov proposed using standard Markov modeling techniques from natural language processing to reduce the password space to be searched, combined with the application of the time-memory trade-off method to analyse the vulnerability of human-memorable passwords. It was shown that 67.6% of the passwords can be successfully recovered using a 2x10^9 search space. However, the limitation of this method is that the passwords were assumed to be human-memorable character-sequence passwords.

In [17], Thing and Ying proposed a new design of an enhanced rainbow table. Maintaining the core functionality of the rainbow tables, the enhanced rainbow table has an improvement of 13% to 19% over the rainbow tables in terms of success rate or an improvement of 50% in terms of storage space.

In this paper, we present an improvement over the method in [17] by describing a way to overcome its main drawback and show that it outperforms the existing rainbow table and the enhanced rainbow table methods.

The rest of the paper is organized as follow. In Section 2, we present a discussion on the time-memory trade-off password recovery methods and how sorting plays an important role in improving the search time. We then give an overview of the sorting method in Section 3. We describe the design of the sorting method in details in Section 4. Analysis and evaluation are presented in Section 5. Conclusions follow in Section 6.

II. PASSWORD RECOVERY AND SORTING TECHNIQUES

The idea of a general time-memory tradeoff was first proposed by Hellman in 1980 [10]. In the context of password recovery, we describe the Hellman algorithm as follows.

We let X be the plaintext password and Y be the corresponding stored hash value of X. Given Y, we need to find X which satisfies h(X) = Y, where h is a known hash function. However, finding X = h^{-1}(Y) is feasibly impossible since hashes are computed using one-way functions, where the reversal function, h^{-1}, is unknown. Hellman suggested taking the plaintext values and applying alternate hashing and reducing, to generate a pre-computed table.

For example, the corresponding 128-bit hash value for a 7-character password (composed from a character set of English alphabets), is obtained by performing the password hashing function on the password. With a reduction function such as $H \ mod \ 26^7$, where H is the hash value converted to its decimal form, the resulting values are distributed in a best-effort uniform manner. For example, if we start with the initial plaintext value of ”abcdefg” and upon hashing, we get a binary output of 0000000....000010000000....01, which is 64 ‘0’s and a ‘1’ followed by 62 ‘0’s and a ‘1’. $H = 2^{63} + 1 = 9223372036854775809$. The reduction function will then convert this value to ”3665127553” which corresponds to a plaintext representation “lwmkgij”, computed from $(11(26^6) + 22(26^5) + 12(26^4) + 10(25^3) + 6(26^2) + 8(26^1) + 9(26^0)$. After a pre-defined number of rounds of hashing and reducing (making up a chain), only the initial and final plaintext values are stored. Therefore, only the “head” and “tail” of a chain are stored in the table. Using different initial plaintexts, the hashing and reducing operations are repeated, to generate a larger table (of increasing rows or chains). A larger table will theoretically contain more pre-computed values (i.e. disregarding hash collisions), thereby increasing the success rate of password recovery, while taking up more storage space. The pre-defined number of rounds of hashing and reducing will also increase the success rate by increasing the length of the “virtual” chain, while bringing about a higher computational overhead.

To recover a plaintext from a given hash, a reduction operation is performed on the hash and a search for a match of the computed plaintext with the final value in the table is conducted. If a match is not found, the hashing and reducing operations are performed on the computed plaintext to arrive at a new plaintext so that another round of search to be made. The maximum number of rounds of hashing, reducing and searching operations is determined by the chain length. If the hash value is found in a particular chain, the values in the chain are then worked out by performing the hashing and reducing functions to arrive at the plaintext giving the specific hash value. Unfortunately, there is a likelihood that chains with different initial values may merge due to collisions. These merges will reduce the number of distinct hash values in the chains and therefore, diminish the rate of successful recovery. The success rate can be increased by using multiple tables with each table using a different reduction function. If we let P(t) be the success rate of using t tables, then $P(t) = 1 - (1 - P(1))^t$, which is an increasing function of t since P(1) is between 0 and 1. Hence, introducing more tables increase the success rate but also cause an increase in both the computational complexity and storage space.

In [18], Rivest suggested a method of using distinguished points as end points for chains. Distinguished points are keys which satisfy a given criteria, e.g. the first or last q bits are all 0. In this method, the chains are not generated with a fixed length but they terminate upon reaching pre-defined distinguished points. This method decreases the number of memory lookups compared to Hellman’s method and is capable of loop detection. If a distinguished point is not obtained after a large finite number of operations, the chain is suspected to contain a loop and is discarded. Therefore, the generated chains are free of loops. One limitation is that the chains will merge if there is a collision within the same table. The variable lengths of the chains will also result in an increase in the number of false alarms. Additional computations are also required to determine if a false alarm has occurred.

In 2003, Oechslin proposed a new table structure [10] to reduce the probability of merging occurrences. These rainbow chains use multiple reduction functions such that there will only be merges if the collisions occur at the same positions in both chains. An experiment was carried out and presented in Oechslin’s paper. It showed that given a set of parameters which is constant in both scenarios, the measured coverage in a single rainbow table is 78.8% compared to the 75.8% from the classical tables of Hellman with distinguished points. In addition, the number of calculations needed to perform the search is reduced as well.

In all the above methods, the stored passwords can be sorted in their alphabetical order. When a password lookup is performed, the time taken to search for this password can therefore be optimized. Hence, the computational complexity to recover the password is low.

In [17], Thing and Ying proposed a new table structure which has an overall improvement over the existing rainbow tables. Even after taking into consideration the effects of key collisions, it was demonstrated that there was a significant increase (between 13% to 19%) in terms of the success rate of recovery, while maintaining the same storage requirement and computational complexity. The novelty of this method lies in the new chain generation process and the removal of the initial hash storage, which resulted in significant storage space conservation (or successful recovery rate improvement).

The main drawback of method is that each password search will incur a significant amount of time complexity. The reason is that the passwords cannot be sorted in the usual alphabetical order now, since in doing so, the information of its corresponding initial hash value will be lost. The lookup will then have to rely on checking every single stored password in the table.

In the following section, we present our proposed sorting method so that password lookup in the stored tables can be optimized.

III. Sorting Method Overview

Based on the method described in [17], we require sorting of the "tail" passwords to achieve a fast lookup. We introduce special characters that can be found on the keyboard (e.g. *, ‘, !, @, :, ”). There are altogether 32 of such non-alpha-numeric printable characters and we assume for now that they do not form any of the character set of the passwords. We insert a number of these special characters into the passwords that we store. The manner in which these special characters are inserted will provide the information on the position of the passwords after the table has been re-arranged in alphabetical order. The consequence is that this will add more storage space compared to [17] but we will illustrate later that the increase in storage space is minimal and is also lesser than the original rainbow table's storage. The advantage of this sorting method is that the passwords can now be sorted and thus a password lookup can be optimized.

As an example, to recover passwords of length 7 consisting of characters in the alpha-numberic character set, and assuming there are 5700 reduction functions and 6.0 x 10^7 chains, a maximum of only 4 special characters are needed in order to span the entire 6.0 x 10^7 passwords. Since the password length is 7, there are 8 different positions where the special characters can be inserted. Hence, the total number of different values which can be obtained by inserting the special characters $>$ 32^4 x (8 + 8x7/2! + 8x7x6/3! + 8x7x6x5/4!) $>$ 6.0 x 10^7. Therefore, only a maximum of 4 characters need to be inserted.

IV. Design of the Sorting Method

In this section, we describe the details of computing and assigning the special characters insertion to perform the sorting, and the derivation of the corresponding initial hash value from the sorted passwords.

Let the 32 special characters be x_1, x_2, ..., x_{32}.

The password with no special character in it has an original position at 0.

If the password is xxxxxxx of length 7, we let $\underline{7}$x$\underline{6}$x$\underline{5}$x$\underline{4}$x$\underline{3}$x $\underline{2}$x$\underline{1}$x$\underline{0}$ be the password with the inserted special characters where the underlined numbers represent the positions of the special characters in the password. More than 1 special character can be assigned to each position. In addition, we define $x_i > x_j$ if the character x_i is to the left of the character x_j in the password.

For passwords with exactly one special character x_i, the original position of the password when x_i is at position a is 32a + i.

For passwords with exactly two special characters x_i, x_j where $x_i > x_j$, the original position of the password when x_i and x_j are at positions a and b respectively is 224 + 32i + j + 512a(a+1) + 1024b

For passwords with exactly three special characters x_i, x_j, x_k where $x_i > x_j > x_k$, the original position of the password when x_i, x_j, x_k are at positions a, b and c respectively is 36064 + 1024i + 32j + k + 16384a(a+1)(a+2)/3 + 16384b(b+1) + 32768c

For passwords with exactly four special characters x_i, x_j, x_k, x_l where $x_i > x_j > x_k > x_l$, the original position of the password when x_i, x_j, x_k, x_l are at positions a, b, c and d respectively is 3935456 + 32768i + 1024j + 32k + l + 131072a(a+1)(a+2)(a+3)/3 + 524288b(b+1)(b+2)/3 + 524288c(c+1) + 1048576d

Note: In the subsequent sections, the same notations as described below will be used.
x_i, x_j, x_k, x_l are the special characters and the values of i, j, k, l range from 1 to 32 inclusive. a, b, c, d are the positions of the special characters and their values ranges from 0 to 7 inclusive.

A. Password Position Assignment

The following describes the procedure of assigning the position of the passwords in the tables to perform sorting.
Step 1: Identify the 32 special characters that do not belong to the character space of the password.

Step 2: Represent each of these 32 characters from x_1 to x_{32}.

Step 3: The first password is left in its original state without any addition of special characters. This will be the password that corresponds to the H value at the start of the chain.

Step 4: The second password will have the character x_1 inserted at the end of the chain. This will be the password that corresponds to the H+1 value at the start of the chain.

Step 5: Subsequent characters are inserted to the passwords such that all the possible characters are inserted to a position. The next higher position will then be allocated for the inclusions of these characters.

Step 6: Once all the positions for 1 character have been filled, 2 characters are used. When they are filled too, 3 characters are used and so on.

Step 7: Continue the assignment of the special characters until all the passwords, excluding the first, have been inserted special characters.

Step 8: These passwords with the addition of special characters can then be sorted in the usual way.

B. Identifying the Positions of Passwords

In the following, we describe the procedure to derive the corresponding initial hash value from the sorted passwords with the inserted special characters.

Step 1: Identify how many special characters are in the password that has been found.

Step 2: Based on the number of special characters and their positions in the password, the corresponding initial hash value is computed as follow.

(a) If there are 0 special characters, then the password corresponds to the initial hash H

(b) If there is 1 special character, then the password corresponds to the initial hash $H + 32a + i$

(c) If there are 2 special characters, then the password corresponds to the initial hash $H + 224 + 32i + j + 512a(a+1) + 1024b$

(d) If there are 3 special characters, then the password corresponds to the initial hash $H + 36064 + 1024i + 32j + k + 16384a(a+1)(a+2)/3 + 16384b(b+1) + 32768c$

(e) If there are 4 special characters, then the password corresponds to the initial hash $H + 3935456 + 32768i + 1024j + 32k + l + 131072a(a+1)(a+2)(a+3)/3 + 524288b(b+1)(b+2)/3 + 524288c(c+1) + 1048576d$

V. Analysis

In this section, we analyse the maximum number of special characters required to sort tables of different sizes and password lengths, as well as demonstrate the storage conservation achieved.

Number of positions that can assigned without using any special character = 1

Number of positions that can assigned using 1 special character = 32 x 8 = 256

Number of positions that can assigned using 2 special characters
= (No. of ways to select 2 special characters) x (No. of ways to place the 2 characters into the 8 positions)
= 32^2 x [8 + 8x7/2] = 36864

In a similar fashion,

Number of positions that can assigned using 3 special characters = 32^3 x [8 + 2x8x7/2 + 8x7x6/3!] = 3932160

Number of positions that can assigned using 4 special characters
= 32^4 x [8 + 3x8x7/2 + 3x8x7x6/3! + 8x7x6x5/4!]
= 346030080

Hence, the total number of positions that can be identified with at most 4 special characters
= 1 + 256 + 36864 + 3932160 + 346030080
= 349999361

Therefore, if less than 350 million passwords are stored, at most 4 special characters are required to identify the position of each password.

Next, we compare the storage requirement between the enhanced rainbow table (incorporating our sorting method) and the original rainbow table by investigating two scenarios.

Scenario 1: 60 million passwords are stored in a rainbow table

Total storage space required for the original table = 1.02 x 10^9 bytes

Total storage space required after the passwords sorting
= 9 x 1 + 10 x 256 + 11 x 36864 + 12 x 3932160 + 13 x 56030719
= 775993340 bytes

Hence, the reduction of storage over the original method
= (1.02 x 10^9 - 775993340) / 1.02 x 10^9
= 0.2392
= 23.92%

Scenario 2: 268 million passwords are stored in a rainbow table

Total storage space required for the original table = 4.556 x 10^9 bytes

Total storage space required after the passwords sorting
= 9 x 1 + 10 x 256 + 11 x 36864 + 12 x 3932160 + 13 x 264030719
= 3479993340 bytes

Hence, the reduction of storage over the original method
= (4.556 x 10^9 - 3479993340) / 4.556 x 10^9
= 0.2362
= 23.62%

We observe that the storage requirement is still significantly reduced even with the inserted characters used in our sorting method.

Tables 1, 2, 3 and 4 show the number of passwords that can be stored in a table with password lengths of 7, 8, 9 and 10 respectively. The values in first row represent the maximum number of special characters added to each password while the values in the second row represent the number of passwords that can be stored.

1	2	3	4
257	37121	3969281	3.50 x 10^8

TABLE 1: PASSWORD LENGTH = 7

1	2	3	4
289	46369	5453089	5.24 x 10^8

TABLE 2: PASSWORD LENGTH = 8

1	2	3	4
321	56320	7208960	7.50 x 10^8

TABLE 3: PASSWORD LENGTH = 9

1	2	3	4
353	67937	9371648	1.05 x 10^9

TABLE 4: PASSWORD LENGTH = 10

We can see that even with only 4 special characters, we are able to store a very large number of passwords in the table. Therefore, a small number of available special characters is sufficient.

Discussions

A shortcoming of this sorting method is the reservation of the special characters, which prevents their usage as password characters. To resolve this, we propose using non-printable characters instead, therefore leaving the printable characters for use as passwords.

VI. Conclusions

This paper describes a sorting mechanism which when applied, has a significant improvement over the orginal rainbow tables. Special characters are added to the storage to allow the sorting of the enhanced rainbow tables so that the password lookup time can be optimized. Even with this insertion of characters to the passwords, the improvement in storage space required to store the same number of passwords is 23% lesser than what is required in the original tables. This is achieved while maintaining the same success rate. Furthermore, it has a speed improvement over the enhanced rainbow tables since it uses a binary search instead of a linear search when performing password lookup. Analysis was also conducted to show that the number of passwords that can be supported by using a small number of special characters, is very large. Therefore, this sorting method can be widely applied. In addition, to circumvent the shortcoming of reserving the printable special characters from being used as passwords, non-printable characters should be chosen for use in this sorting method instead.

References

[1] S. M. Smyth, “Searches of computers and computer data at the United States border: The need for a new framework following United States V. Arnold”, Journal of Law, Technology and Policy, Vol. 2009, No. 1, pp. 69-105, February 2009.

[2] Google News, “Favorite passwords: ‘1234’ and ‘password”’, http://www.google.com/hostednews/afp/article/ALeqM5jeUc6 Bblnd0M19WVQWvjS6D2puvw, [retrieved, December 2009].

[3] Cain and Abel, “Password recovery tool”, http://www.oxid.it, [retrieved, December 2010].

[4] John The Ripper, “Password cracker”, http://www.openwall.com, [retrieved, December 2010].

[5] LCPSoft, “Lcpsoft programs”, http://www.lcpsoft.com, [retrieved, December 2010].

[6] S. Contini, and Y. L. Yin, “Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions”, Annual International Conference on the Theory and Application of Cryptology and Information Security (AsiaCrypt), Lecture Notes in Computer Science, Vol. 4284, pp. 37-53, 2006.

[7] P. A. Fouque, G. Leurent, and P. Q. Nguyen, “Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5”, Advances in Cryptology, Lecture Notes in Computer Science, Vol. 4622, pp. 13-30, Springer, 2007.

[8] Y. Sasaki, G. Yamamoto, and K. Aoki, “Practical password recovery on an MD5 challenge and response”, Cryptology ePrint Archive, Report 2007/101, April 2008.

[9] Y. Sasaki, L. Wang, K. Ohta, and N. Kunihiro, “Security of MD5 challenge and response: Extension of APOP password recovery attack”, The Cryptographers’ Track at the RSA Conference on Topics in Cryptology, Vol. 4964, pp. 1-18, April 2008.

[10] M. E. Hellman, “A cryptanalytic time-memory trade-off”, IEEE Transactions on Information Theory, Vol. IT-26, No. 4, pp. 401-406, July 1980.

[11] D. Todorov, “Mechanics of user identification and authentication: Fundamentals of identity management”, Auerbach Publications, Taylor and Francis Group, June 2007.

[12] R. Rivest, “The MD5 message-digest algorithm”, IETF RFC 1321, April 1992.

[13] National Institute of Standards and Technology (NIST), “Secure hash standard”, Federal Information Processing Standards Publication 180-2, August 2002.

[14] H. Dobbertin, A. Bosselaers, and B. Preneel, “Ripemd-160: A strengthened version of RIPEMD”, International Workshop on Fast Software Encryption, Lecture Notes in Computer Science,

Vol. 1039, pp. 71-82, Springer, April 1996.

[15] P. Oechslin, “Making a faster cryptanalytic time-memory trade-off”, Annual International Cryptology Conference (CRYPTO), Advances in Cryptography, Lecture Notes in Computer Science, Vol. 279, pp. 617-630, October 2003.

[16] A. Narayanan, and V. Shmatikov, “Fast dictionary attacks on passwords using time-space tradeoff”, ACM Conference on Computer and Communications Security, pp. 364-372, 2005.

[17] V. L. L. Thing, and H. M. Ying, “A novel time-memory trade-off method for password recovery”, Digital Investigation, International Journal of Digital Forensics and Incident Response, Elsevier, Vol. 6, Supplement, pp. S114-S120, September 2009

[18] D. E. R. Denning, “Cryptography and data security”, Addison-Wesley Publication, 1982.