This full-text paper was peer-reviewed and accepted to be presented at the IEEE WiSPNET 2016 conference.

Securing Passwords from Dictionary Attack with Character-Tree

Jacob Jose, Tibin T. Tomy, Vibin Karunakaran
Department of Information Technology,
College of Engineering Kidangoor,
Kottayam, India.
jacobjose123@gmail.com

Anjali Krishna V, Anoop Varkey, Nisha C.A.
Department of Information Technology,
College of Engineering Kidangoor,
Kottayam, India.
anjalikrishnav945@gmail.com

***Abstract*—Most websites use passwords for authenticating user identity and for allowing access to website resources that may contain sensitive information. A large number of people use dictionary words for creating passwords. These user passwords are subjected to one-way hash functions and are stored inside the database as corresponding hash values instead of plaintext. A potential hacker can use brute-force, rainbow table or dictionary attacks to get the input password from the hash values and the most reported real life hacks were done by cracking password hashes using dictionary attack. Currently, users are allowed to register in websites only with passwords that obey the security policies. It is noted that, even though passwords with certain patterns are accepted as strong by the existing policies, they are vulnerable for a dictionary attack based on those patterns. This paper proposes a novel method for ensuring security for passwords against such dictionary attacks. This method, checks strength of the user passwords using a dictionary which is stored as a character tree. This system helps to create strong password hashes that are resistant to dictionary attacks. This approach thus offers advanced and superior protection for passwords from cracking attempts.**

Index Terms*—*authentication, character-tree, dictionary attack, password hashes, hash-cracking, security, website-hacks.

I. Introduction

Security is a major area of concern when related to information on web.Many website databases contain details of millions of users. Hence it is vital for any website to ensure security for all user data stored in it. Also, it is essential to validate the identity of users before retrieving data and services of a web application software. Authentication is thus a requirement for information security [1, 2]. Various mechanisms are being used for authentication according to the convenience of users such as security hardware tokens, PINs, biometric fingerprints, passwords etc [3]. Among these existing methods, the most commonly used one is the password based authentication mechanism due to its simplicity in implementation when compared to other methods [3, 4]. For ensuring security, most developers store passwords within databases as their salted hash values instead of plaintext [5, 6]. It is possible for hackers to gain access into database tables and their entries through SQL injection vulnerabilities [7, 8]. A hacker who retrieves user passwords from the database gains unauthorized access to user accounts and other sensitive information stored in the hacked website [9]. Recently, many popular websites were victims of such attacks and password leaks [10].

Even though, passwords chosen by any user during the registration process are subjected to one-way hashing functions to produce a hash value, attackers can gain actual input passwords from them [11]. Hackers use brute-force, dictionary and rainbow table attacks to retrieve the input password plaintext from its hash value [12]. Considering these facts, it is essential to pay great importance for the security of user passwords.

Most web applications now insists users to add digits, symbols and special characters along with their passwords to make it more strong and complex [3, 13]. For example, a dot (“.”), an exclamation mark (“!”) or digits “0”, “1” etc.are often appended at the end of a password by the users to satisfy the system requirements. Likewise, some users create strong and long passwords by repeating a dictionary word two or three times (eg.helphelp, helphelphelp) [13, 14]. Also, certain password generators provide pattern based passwords which they claim to be strong and complex. It is possible for hackers to identify the common password patterns and generate a new pattern based dictionary file from an ordinary dictionary file [14]. In such cases, these pattern based passwords are not capable of resisting attacks and are vulnerable to such threats. Hence, even a very strong hash function cannot prevent hash attacks against a weak password. This paper focuses on securing passwords from a pattern based dictionary attack.

This paper is organized as follows: Section II analyzes common passwords and attacks in detail. It discusses dictionary attacks, existing and proposed security methods. Section III explains pattern based dictionary attack. Implementation of this system is explained in section IV and the test results are discussed in section V. Section VI concludes the paper.

II. The Analysis

Many popular websites were hacked recently which resulted in leakage of millions of user passwords [10]. The website hacks and data leakages that occurred these days shows the severe threat faced by information on the web. Most popular auction website eBay revealed that hackers had managed to steal personal records of its 233 million users in May 2014. The critical attack leaked private user credentials like usernames, passwords, phone numbers and physical addresses compromised and managed to gain access to sensitive data [15].

978-1-4673-9338-6/16/$31.00

Another major attack was on the popular website rockyou.com which led to exposure of its 32.6 million user passwords through SQL injection vulnerability in December 2009 [16]. Security analysts and researchers had the leaked password list of rockyou published on web as a great resource for analyzing passwords [14, 16]. It becomes clear that, a leaked password list can serve as the best source file for a dictionary attack.

On analyzing leaked passwords, it is identified that the most commonly used passwords are simple dictionary words and the most used method to crack password hashes is the dictionary attack [17, 18, 19]. People prefer creating passwords that are easy to remember and thus they choose common dictionary words [20].

A. Dictionary Attack

Dictionary attack seems to be the most used method for cracking password hashes. Since password analysis shows a high frequency usage of dictionary words, a dictionary attack gives much fast and successful results while cracking hashes [14, 20, 21]. It uses very large dictionary files containing millions of possible passwords or combination of dictionary words. The hash value of each password in the dictionary file is calculated and compared with an input hash value of any unknown user password. A match found reveals the plaintext in dictionary corresponding to the hash value which will be same as the input password [17, 21]. This is a fast method when compared to other methods for attack,but have a success rate lower than brute-force attack.Observing the reported attacks and common passwords, it can be assumed that performing a dictionary attack gives good success rates.Hence now,most of the attackers use this method to crack a large number of passwords [21, 22].

B. Existing Security Scheme

Assumptions were made that,passwords are secure and safe if they are made resistant to dictionary attacks. Presently,most of the websites do not support simple dictionary words to be used as passwords while registering new accounts. This is meant for security of user passwords,since reported dictionary attacks easily crack such simple passwords. Users are advised to insert symbols or digits along with simple words to make their password a strong one. It is observed that most passwords used now are a combination of dictionary words, digits and some symbols. Such a password with certain pattern is accepted as strong by the present password policies [23, 24].

The identified common pattern combinations can therefore be dual or triple combination of various character groups having variable length. The character groups identified in common from a leaked password file can be analyzed as [alpha]: any alphabet from a to z and from A to Z, [digit]: any digit from 0 to 9, [symbol]: any of the following characters: . , ? ! ; : # $ % & ( ) * + - / < > = @ [ ] ^{}| . Many passwords can have common patterns and, they are all vulnerable for cracks by a pattern based attack. On analyzing and identifying common patterns, regular expressions can be used to generate passwords with patterns. Such generated passwords if used in a dictionary file for guessing passwords, will lead to a successful method for cracking the passwords currently accepted as strong [25]. Considering the possibility for any hacker to perform an attack using an improved pattern based dictionary file, security of these pattern based passwords can no longer be guaranteed [14].

III. PATTERN BASED DICTIONARY ATTACK

A pattern based dictionary attack can be otherwise called as an improved dictionary attack. The key factor which makes it different from ordinary dictionary attack is the basic dictionary file used for attack. An ordinary dictionary attack uses a very large base file containing millions of possible passwords which are simple dictionary words. A pattern based dictionary attack uses a huge sized file containing more passwords than in a simple dictionary file. This file is generated by adding identified common patterns to the passwords in the simple dictionary file. Some of the identified common password patterns are: Appending, Prefixing, Inserting, Repeating, Sequencing, Replacing, Reversing, Capitalizing, Special-format and Mixed patterns. A software tool called pattern based password generator constructed to generate a pattern based dictionary file by implementing identified patterns can efficiently crack password hashes [14]. Considering the fact that it is possible to perform a dictionary attack with patterns, it is essential to develop security measures against such a threat.

A. Proposed Security Scheme

The proposed system does not allow even using a pattern based password which includes a dictionary word. Considering the researches on a pattern based dictionary attack, a password obeying the existing password policy which contains a dictionary word can never be accepted as safe. Hence, here when a user submits a password with certain pattern, it is subjected to pattern based computations to detect any dictionary word in it. If any dictionary word is found in the input, it will be shown as a weak password and the user is therefore not allowed to use it. Here, the complexity rules are much stronger so as to insist users to submit passwords containing words or characters other than dictionary words. Even when a pattern based dictionary attack is made, none of the passwords registered according to this protocol will be cracked.

IV. IMPLEMENTATION

A character-tree structure is used here for storing passwords in a dictionary file. Most commonly used pattern types are identified from the input passwords and it is checked whether it is a dictionary word or not. A search in the character tree finds a word if the input password contain any dictionary words. The character tree gets updated by adding words that are newly recognized. This tree storage gives same performance as that of a trie tree data structure for storing words [26].

A. Dictionary File Storage

The character tree has it's each node saved as a file to store passwords from a dictionary file. Each node is saved with the character of the password being read as the name and name of the child files as its content. A character tree-structured dictionary store is illustrated in Figure 1. The root node of the tree has the alphabets from a to z, numbers from 0 to 9 and the special characters . , ? ! ; : # $ % & ( ) * + - / < > = @ [ ] ^{}| [] as child files. An example of storing words in the tree is shown in Figure 2.

The process of storing the passwords in a tree structure is proposed by the following algorithm.

Input:A dictionary file containing passwords in
Output:A tree structured password dictionary file out
Initialisation :i=0,countname=0

1: Read first line of the input into a variable "input".
2: Compute "input" length into a variable "length".
3: Open file "rootfile.txt"
4: "filename"=rootfile.txt
LOOP Process
5: **for** $i < length$ **do**
6: Open "filename"
7: Skip first line in file
8: (First line is a flag indicating end of a word.)
9: **if** ($input.charAt(i)$==$first\ character\ of\ any\ line\ in\ the\ file$) **then**
10: filename=that line in the file
11: i++
12: **else**
13: Write to the file(input.charAt(i)+countname+.txt)
14: filename=(input.charAt(i)+countname+.txt)
15: Create a new file(input.charAt(i)+countname+.txt)
16: Write "0" to file
17: countname++
18: i++
19: **end if**
20: **end for**
21: Open "filename"
22: Replace first line with 1

The above shown algorithm maps each line of the input file into nodes of the tree and the process repeates till end of the file. First line of every file is a flag having value either 0 or 1. A flag value of 1 indicates end of a word in the dictionary tree and a file with flag 0 shows that the current node is not the end of a word. A search function in the tree has to find the words from the nodes it traversed. Therefore, assigning flags to every node will help to identify a complete word in the tree. Each file name is a string obtained by concatenating the character at i^{th} position of the word and the variable countname. Here, countname is an incrementing integer variable that gives a unique identity to each file. Example: Considering the words "agree" and "agreement" in the tree, the character "e" has to be stored as the child of "r", "m" and "e" .Here, the character "e" requires an identity in each case as it repeats in a word.

TABLE I
NORMAL PATTERN COMPUTATIONS

Input	Output
passion	passion

Thus, countname provides an identity to a same character at different nodes in the tree.

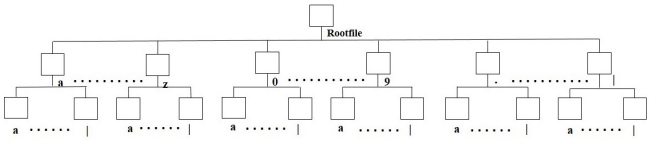

Fig. 1. An illustration of Tree Storage

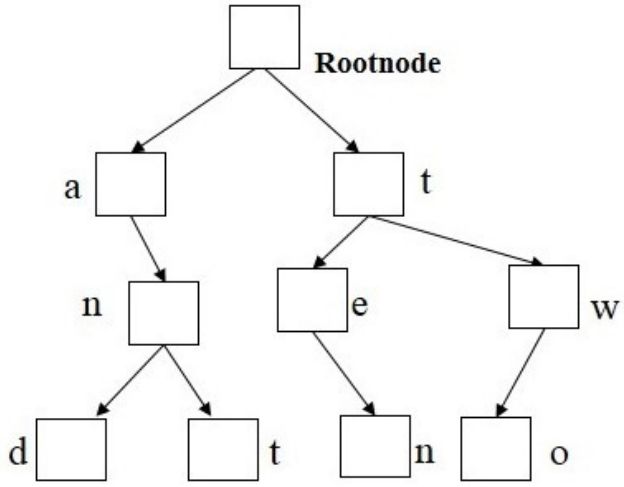

Fig. 2. An illustration that shows storing the words "and","ant","ten" and "two" in tree

B. Computing Password Pattern

Whenever any user enters a password to the system, it is read as input and 7 process threads perform pattern computations on the input password. The seven threads run simultaneously and any one of them identifies the correct pattern, while others return false on completion of the process.

1) Normal Pattern: If any input password received by the system is a simple dictionary word, (eg: passion) it is then assumed to have this pattern type. This pattern type can otherwise be defined as null pattern type, since it actually possess no patterns. A thread assigned for this type of pattern, after receiving a simple word as input quickly finds it and returns a true statement to the system. The system do not allow this input type to be used as password. The computations performed for the input password of this pattern type are shown in Table I.

TABLE II
PREFIXING PATTERN EXAMPLES

Pattern Example	Password Examples
Prefixing [0-9]	1password, 1bitch, 1lover
Prefixing 123	123abc, 123qwe, 123asd
Prefixing # 1	#1bitch, #1pimp, #1abcd
Prefixing !	!password, !iloveyou, !basketball)
Prefixing *dot*	.password, .iloveyou, .ghgh, .followingyou

TABLE III
PREFIXING PATTERN COMPUTATIONS

Input	Output
123passion	23passion
23passion	3passion
3passion	passion

2) Prefixing Pattern: Passwords of this pattern type can be defined as a combination of character groups, generally of the form [digit] + [alpha]. This is formed by prefixing a certain digit or any special character/digit or character groups at the beginning of a dictionary word. Some common prefixing pattern types and example passwords are shown in Table II [14]. The computations made on passwords of this pattern type are depicted in Table III. The prefixing thread performs computations on the input password and generates an output. If the input password has a prefixing pattern, the thread will identify a dictionary word from the password by searching in the tree store and otherwise, it returns a false statement.

3) Appending Pattern: The passwords of this type are formed as a combination of character groups [alpha] + [digit]. Here, a certain digit or special character/digit or character groups are added at the end of a dictionary word. Table IV shows passwords of this type [14]. A thread for identifying this pattern type executes pattern based computations on the input through some steps as shown in Table V.

TABLE IV
APPENDING PATTERN EXAMPLES

Pattern Example	Password Examples
Appending [0-9]	password1, princess1, angel1
Appending 123	abc123, love123, red123
Appending 1234	abcd1234, abc1234, love1234
Appending 101	love101, zoey101,) sweet101
Appending *dot*	password., iloveyou., followingyou.
Appending !	iloveyou! , password! , rockyou!

TABLE V
APPENDING PATTERN COMPUTATIONS

Input	Output
passion123	passion12
passion12	passion1
passion1	passion

TABLE VI
REPEATING PATTERN EXAMPLES

Pattern Example	Password Examples
Repeating number "N" for N times	1, 22, 333, 4444, 55555, 666666, 7777777, 88888888, 999999999
Repeating number groups	123123, 303030, 292929, 420420, 007007, 789789, 123456123456
Repeating [0-9] numbers	111111, 11111, 11111111, 222222, 22222 333333
Repeating birth years	19871987, 19891989, 19921992, 19861986, 19931993
Repeating words	lovelove, catcat, kisskiss, oneone, twotwo, passwordpassword, usausa, blablabla
Repeating letter groups	abcabc, abcabcabc, ABCABC, defdef, defdefdef
Repeating [a-z]	aaaaaa, aaaaa, bbbbbb , bbbbb, bbbbbbbb, bbbbbbb
Repeating symbols	....., ..., ...

4) Repeating Pattern: This type of password pattern is formed by repeating a dictionary word two or three times. Here, it includes only one character group, [alpha] for forming the pattern. Common repeating pattern types and some password examples are shown in Table VI [14]. In case of this pattern type, it do not require a separate process thread. Instead either the prefixing or appending pattern processing threads will find out a dictionary word in this password pattern. Therefore, replacing pattern type can be processed in either prefixing or appending type of patterns. The method of processing this pattern type is described using Table VII and Table VIII.

TABLE VII
REPEATING PATTERN COMPUTATIONS BY PREFIXING

Input	Output
lovelove	ovelove
ovelove	velove
velove	elove
elove	love

TABLE VIII
REPEATING PATTERN COMPUTATIONS BY APPENDING

Input	Output
lovelove	lovelov
lovelov	lovelo
lovelo	lovel
lovel	love

TABLE IX
REPLACING PATTERN EXAMPLES

Replaced Letter	Replaced with	Password Examples
a	4	d4niel, c4r0lin4, dr4gon
a	@	p@ssword, t@ylor, f@mily
b	6	straw6erry, septem6er, remem6er
e	3	monk3y, socc3r, princ3ss
g	6	soccer6irl, hun6ry, ran6ers
g	9	an9els, en9ine, dan9er
i	1	* 1loveyou, , mel1ssa, stup1d
i	!	pr!ncess, sunsh!ne, pr!nce,
i	|	M|ChE|Le, m|r@cleS, sl|ther
l	1	P1ayer, ash1ey, ye11ow,
s	5	pas5word, chee5e, augu5t,
s	$	$prite, be$tfriend, $pecial,

TABLE X
REPLACING PATTERN COMPUTATIONS

Input	Output
p4s5i0n	p4s5i0n
p4s5i0n	pas5i0n
pas5i0n	pas5i0n
pas5i0n	passi0n
passi0n	passi0n
passi0n	passion

5) Replacing Pattern: Passwords of this pattern type are obtained by replacing certain alphabets with digits or special characters. Passwords shown in Table IX are formed by this pattern type [14]. If the password entered to the system is of this pattern type,the replacing process thread finds a successful result at the end. The process is explained in Table X.

6) Reversing Pattern: This type of pattern creates passwords by reversing the order of characters in a dictionary word. Some examples of this pattern type are drowssap, uoykcor, fedcba, elgoog, uoyevoli, ssecnirp, yraunaj, ylevol [14]. Here, an input password is processed in various steps to reverse its order till it finds out a dictionary word. This process of finding a dictionary word by reversing the input is illustrated in Table XI.

7) Capitalizing Pattern: By this pattern some users create passwords by converting any lower-case letters of a dictionary word into upper-case. Examples of passwords of this pattern type are shown in table XII [14]. Here, the capitalizing thread first checks for the three common pattern types in the input. At first, it checks for an upper-case letter at both the first and last positions of the input. If the pattern is not obtained, the process thread then checks for an upper-case letter at first and last positions of the input. If any of these pattern is identified, the upper-case letters are exchanged with their lower-case equivalents. As an example, the passwords PassioN, Passion and passioN can be converted into passion. Examples of this pattern type computations are given in Table XIII.

TABLE XI
REVERSING PATTERN COMPUTATIONS

Input	Output
noissap	pnoissa
pnoissa	panoiss
panoiss	pasnois
pasnois	passnoi
passnoi	passino
passino	passion

TABLE XII
CAPITALIZING PATTERN EXAMPLES

Pattern Example	Password Examples
Capitalization of 1st letter	Password, Princess, Jessica, Michael, Nicole, Daniel
Capitalization of last letter	rockU, passworD, whoamI, princesS
Capitalization of both first and last letters	LiverpooL, DaniellE, MichellE,RockU LoveyoU,MonkeY

TABLE XIII
CAPITALIZING PATTERN COMPUTATIONS

Input	Output
PassioN	passion
Passion	passion
passioN	passion

C. Searching in Tree

Each of the process threads produces outputs at each step of its computation. Every generated output is subjected for a search operation in the tree store to check whether it is a dictionary word or not. While searching, any one of the process thread successfully finds a dictionary word in the output and then the process of search is terminated by returning a result to the system. This process can be better illustrated using Table XIV. Since the search is made in a tree based file store, the process becomes efficient and yields a quick result. The process of searching a given password in the dictionary tree is explained by the following algorithm.

Input:A password in
Output:Existence of the password in dictionary out
Initialisation :i=0
1: Read the input password into a variable "input".
2: Compute "input" length into a variable "length".
3: Initialize the file "rootfile.txt"
LOOP Process
4: **for** $i < length$ **do**
5: Open "file"
6: Read lines in the "file"

TABLE XIV
SEARCHING IN DICTIONARY

Input/Password	Output/Keyword	Search Result
123passion	23passion	Not Found
23passion	3passion	Not Found
3passion	passion	Found

if (*input.charAt*(i)==*first letter of the line read from the file*) **then**
  file=that line in the file
  i++
else
  Return false
end if
end for
if (*First line of the file*==1) **then**
  Return true
else
  Return false
end if

A password entered is taken as input for searching in the dictionary tree. The character at i^{th} position of the input is compared with the first character of any line read from the file. If they match, the file is opened and first line in the file is read by the process. The first line of every file contains the flag bit. If it is 0, value of "i" is incremented till it becomes equal to the value of length and the process repeats until a flag of 1 is read from any file. Example: Both the words "ant" and "antenna" are stored in the tree as shown in the Figure 3. Here, while searching for the word antenna, the process thread identifies the word ant first. Since the pointer at the input moves to next character, search in the tree proceeds to next node and finds the word antenna in it.

On the other case, if the input word is ant, the search function finds it in the tree and since pointer at input reaches end of the word, process terminates by returning a found result. If a 0 is read from first line of any file, the process returns a false indicating that the input word is not found in the dictionary and continues search till end of the input word. Otherwise, if a 1 is read from any file, the process returns a true showing that the input word is found in the dictionary. The process of traversing the dictionary tree and finding an input word in it is illustrated in Figure 4. Here, the word "abacus" exists in the tree and a search function finds it. If an input "abfcus" is given to the system, the word does not exist in the tree as the node "b" do not have a child node "f". Thus, the search function will not find the input word by traversing the tree. Hence, it returns a Not Found result.

D. Pattern Recognition

A most significant functionality offered by this character tree is that, it identifies new passwords entered by any user and adds them to the tree so as to update itself. This function of identifying new passwords and patterns from an input is shown by the algorithm given below.

Input:A password in
Output:Words and Characters in the password out
Initialisation :a=0,b=0,f=0,i=0,k=0,
Read the input password into a variable "password".
Compute "password" length into a variable “length”.
Initialize arrays prefix[],append[],lines[]
LOOP Process
j=length

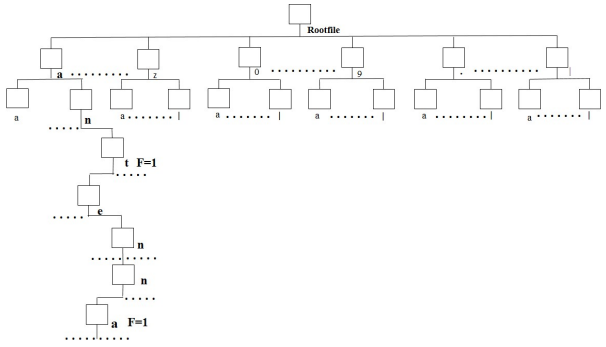

Fig. 3. An illustration that shows identifying words "ant" and "antenna" in the tree

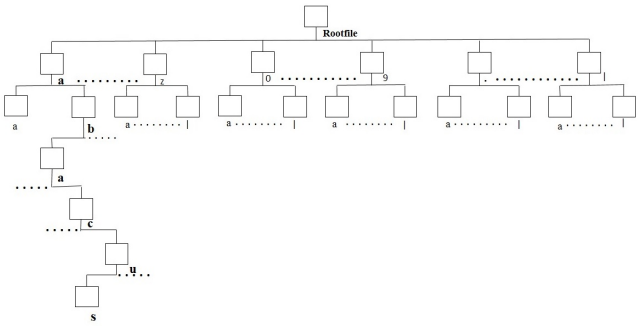

Fig. 4. An illustration that shows identifying word "abacus" in tree

Generate a substring from "password" with starting index "i" and ending index "j".
Search the substring in dictionary.(This dictionary contains the common words used in English language.)
if (*Found*) **then**
  Set f=1
  prefix[a++]=i
  append[b++]=j
  i=j+1
  lines[k++]=substring
  Go to step 17
end if
f=0
j—
if $j > i$ **then**
  Go to step 5
end if
if f=0 **then**
  i++
end if
if $i <$ *length*) **then**
  Go to step 4.
  Concatenate string in array lines[](which are the words in the input password.)
  Last element in array append[] is the starting index of appending string.

27: First element in array prefix[] is the ending index of prefixing string.
28: **end if**

If an input word is found to be strong and not existing in the current dictionary tree, system searches for it in a separate dictionary file containing more words by splitting the input word into substrings. This substrings are then stored in a file as objects with a count value as its parameter. Then the prefixing and appending characters are identified and stored separately in files. Now, value of count increments if the word is identified again in the system. The word is stored into the dictionary tree storage as a new node when the value of count exceeds the defined value. The characters identified and stored in files are also added to the table of patterns so as to update the pattern tables. Otherwise, if a new password identified is not found in the dictionary file, the word itself is stored in a file as an object as in the above case. The word is added into the dictionary tree when the count value exceeds its defined value as mentioned earlier. Thus, the dictionary tree updates itself by adding new passwords and the next time, system performs a search in this updated dictionary tree.

V. TEST RESULTS

Performing tests on checking password strength with the dictionary tree provided fast results. It took only an approximate of 41 hours to map the dictionary file into a tree-structured storage with the proposed algorithm. Also, searching in the tree store produced results within a time bound of 1 to 2 seconds. The system gives accurate results on the strength of an input password based on the theory of avoiding dictionary words.

VI. CONCLUSION

Authentication systems face critical threats due to weak passwords. Hacking websites by seizing unsalted password hashes has become possible to a great extent through dictionary attacks.

This paper proposes a security system developed in java as a web application that can be implemented on any website or any platform. On analyzing common password patterns, the possibility for an attack based on common pattern based passwords were identified. It is observed that for ensuring security, it is required to avoid all dictionary words from passwords.Therefore, this system is developed so as to offer protection for passwords from dictionary attack by avoiding all common dictionary words.

This system use a novel method of storing a very big password file in a tree structured format. With the help of 7 parallel running threads, password patterns are identified and an efficient search in the tree finds dictionary words in an input. There are possibilities for attackers to develop a pattern based dictionary and perform a pattern based attack on passwords. From this observation, this proposed system offers protection from such an improved dictionary attack and can be considered to be a next generation password security scheme. Also, this system if implemented on any confidential website, can safeguard their sensitive information from any improved dictionary attacks and it offers an advanced security over the existing system.

VII. ACKNOWLEDGEMENT

The authors would like to thank TEQIP phase II for providing financial support to implement this project and draft this article.

REFERENCES

[1] Qiang Wang, Zhiguang Qin, “Stronger user authentication for web browser,” *Advanced Computer Theory and Engineering (ICACTE) 2010 3rd International Conference.* Volume:5, Aug 2010.
[2] http://www.infosec.gov.hk/english/technical/files/web-app.pdf.
[3] L.O’Gorman, “Comparing passwords,tokens and biometrics for user authentication,” *Proceedings of the IEEE* vol. 91, no. 12, pp. 2021–2040.
[4] Shuo Zhai, "Design and implementation of password-based identity authentication system," *Computer Application and System Modeling (ICCASM), 2010 International Conference* Volume:9 22-24 Oct. 2010.
[5] https://www.youtube.com/watch?v=8ZtInClXe1Q.
[6] https://www.youtube.com/watch?v=b4b8ktEV4Bg.
[7] https://www.youtube.com/watch?v=-jKylhJtPmI.
[8] A. Sadeghian, M. Zamani, S. M. Abdullah, "A Taxonomy of SQL Injection Attacks," *Informatics and Creative Multimedia (ICICM), 2013 International Conference* 4-6 Sept. 2013.
[9] https://www.youtube.com/watch?v=BcDZS7iYNsA.
[10] http://www.forbes.com/sites/thomasbrewster/2015/10/28/000webhost-database-leak/#2715e4857a0b6c86e6ae17c1.
[11] G. Martinovic, L. Horvat, J. Balen, "Stochastic approach on hash cracking," *MIPRO, 2012 Proceedings of the 35th International Convention.*
[12] H. Kumar, "Rainbow table to crack password using MD5 hashing algorithm," *Information & Communication Technologies (ICT), 2013 IEEE Conference*11-12 April 2013.
[13] https://www.schneier.com/blog/archives/2014/03/choosing-secure-1.html.
[14] Emin Islam Tath, "Cracking more password hashes with patterns," *IEEE Transactions on Information Forensics and Security* Volume:10.
[15] http://www.cnet.com/news/ebay-hacked-requests-all-users-change-passwords/.
[16] http://www.scmagazine.com/rockyou-hack-compromises-32-million-passwords/article/159676/.
[17] http://blog.dictionary.com/dictionaryattack/.
[18] https://www.hackthissite.org/articles/read/1112.
[19] http://web.cs.du.edu/ mitchell/forensics/information/pass-crack.html.
[20] https://wiki.skullsecurity.org/Passwords.
[21] Vishwakarma D., Madhavan C.E.V, "Efficient dictionary for salted password analysis," *Electronics, Computing and Communication Technologies (IEEE CONECCT), 2014 IEEE International Conference* Jan 2014.
[22] https://www.youtube.com/watch?v=2hveQ8QZ9MQ
[23] Saikat Chakrabarti, Mukesh Singhal, "Password Based Authentication: Preventing Dictionary Attacks,"*Published by the IEEE Computer Society* Issue No.06 - June (2007 vol.40), pp: 68-74.
[24] https://support.google.com/a/answer/33386?hl=en.
[25] Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B. "Password Cracking Using Probabilistic Context-Free Grammars"*Security and Privacy, 2009 30th IEEE Symposium* 17-20 May 2009.
[26] http://software.ucv.ro/ cmihaescu/ro/laboratoare/SDA/docs/trie.pdf.