International Journal of Engineering Research and Advanced Technology (IJERAT)

DOI: http://dx.doi.org/10.7324/IJERAT.2018.3218

E-ISSN : 2454-6135

Volume.4, Issue 4

April -2018

Honeywords: Making Password Cracking Detectable

Shubham Sawant[1], Pratik Saptal[2], Kritish Lokhande[3], Karan Gadhave[4] and Prof. Randeep Kaur[5]

Department of Computer Engineering,

Terna Engineering College,

Mumbai University

Nerul, Navi Mumbai

India

ABSTRACT

As society is dependent on digital world, the threat continues to rapidly increase. Every year new mechanism against cyber security threats is introduced. At same time the cybercriminals additionally create new techniques to beat these efforts. One in every of the vital security issue is with disclosure of password file. To tune up this issue the concept of honeywords i.e. false password is introduced. We tend to show a straightforward methodology for raising the protection of hashed passwords: the upkeep of additional "honeywords" (false passwords) related to every user's account. An adversary who steals a file of hashed passwords and inverts the hash function cannot tell if he has found the password or a honeyword. The tried use of a honeyword for login sets off an alarm. An auxiliary server (the "honeychecker") will distinguish the user password from honeywords for the login routine, and will set off an alarm if a honeyword is submitted.

Keywords: *Honeywords, Honeychecker, Passwords, cyber security.*

I. INTRODUCTION

Information security has become a most prominent requirement in this era which is done using some authentication method. Many different methods for authentication exists (e.g. Patterns, Passwords etc.). Now-a-days most generally used method for authentication is passwords. Security of password is an important issue. A password is a secret word, which a user must input during a login, only after that it is possible to get access. In password based systems, developers must take care that passwords should not be stored in databases in plaintext or with unsalted hash values. In past, many hacking attempts had made possible for attackers to gain unauthorized access to the sensitive data as well as user passwords stored in database. The mechanism of Password protection helps us to protect information from unauthorized users. An adversary who has stolen a file of hashed passwords can often use brute-force search to find a password p whose hash value H(p) equals the hash value stored for a given user's password, thus allowing the adversary to impersonate the user. A recent report by Mandiant1 illustrates the significance of cracking hashed passwords in the current threat environment. Password cracking was instrumental, for instance, in a recent cyberespionage campaign against the New York Times. The past year has also seen numerous high profile thefts of files containing consumers' passwords; the hashed passwords of Evernote's 50 million users were exposed as were those of users at Yahoo, LinkedIn, and eHarmony, among others. The LinkedIn passwords were using the SHA-1 algorithm without a salt and similarly the passwords in the eHarmony system were also stored using unsalted MD5 hashes . Indeed, once a password file is stolen, by using the password cracking techniques like the algorithm of Weir et al. It is easy to capture most of the plaintext passwords. Herley and Florencio proposed a new approach to detect the malicious behaviour on every incorrect or unauthorized login. For every single user false login attempts with few passwords will generate honeypot accounts (fake accounts) so that malign behaviour is caught. Recently, Juels and Rivest have presented the honeyword mechanism to detect an adversary who attempts to login with cracked passwords. The concept is that for each username they build a set of sweetwords in which one word is the real password and the others are honeywords (false passwords). This approach was proposed considering the second issue about detecting the malicious entry. When an adversary tries to get access using any of the honeyword an alarm is triggered which notifies the administrator about the password file breach. This approach is not terribly deep, but it should be quite effective, as it puts the adversary at risk of being detected with every attempted login using a password obtained by brute-force solving a hashed password. Consequently, honeywords can provide a very useful layer of defense. In any case, our hope is that this paper will help to encourage the use of honeywords.

2. LITERATURE SURVEY

PC users and businesses use computers for numerous functions, including transferring and storing digital data. Users may wish to keep this information restricted for a number of reasons. User authentication can be used for this purpose. User authentication is a means of verifying, the identity of an individual requesting access to certain information. There are several ways user authentication can be verified. The user can confirm his identity via secret information the requester uniquely knows, such as a password. So, Password Security is an important aspect. Compared with traditional password based authentication methods, Honeyword based authentication method has shown better results.

2.1 The Dangers of Weak Hashes

K. Brown in 2012 discussed that by implementing few best
practices, the damage caused by password leak can be minimized. It was found that there had been several publicity password leaks in June 2012 including LinkedIn, Yahoo, and eHarmony. Secure system should not have any vulnerabilities that will allow hackers to get access to password files and should make sure that if the password hashes are been hacked it should not be easy to generate passwords from the hashes. Thus these leaks have proved that many companies are following weak hashing mechanisms. In this paper, we have discussed about basics of password hashing and best practices that should be followed while password storage .

2.2 Use of Honeywords

Juels and Rivest proposed a method for improving the security of hashed passwords. To improve the security of the hashed password, honeywords (false passwords) needs to be generated for each user account. Attacker who steals a file of hashed passwords and inverts the hash function cannot tell if he has found the password or a honeyword. If the attacker attempts to login with the use of honeyword the auxiliary server will set off an alarm. When honeywords are used a successful brute-force password break does not give the adversary confidence that he can log in successfully without detection. Proposed system includes honeyword generation algorithms and comparison with different factors.

2.3 Examination of a New Defense Mechanism: Honeywords.

It has become much easier to crack a password hash with the
advancements in graphical processing unit technology. Once the password has been recovered no server can detect any illegitimate user authentication (if there is no extra mechanism). They propose an approach for user authentication, in which some false passwords, i.e., “honeywords” are added into a password file, in order to detect impersonation. The authors in propose an interesting defense mechanism under a very common attack scenario where an adversary steals the file of password hashes and inverts most or many of the hashes. The honeyword system is powerful defense mechanism in this scenario. Namely, even if the adversary has broken all the hashes in the password file, he cannot login to the system without a high risk of being detected. Hacking the honeychecker has also no benefit to the
adversary since there is no information about a user’s password or honeyword in the honeychecker .

2.4 Achieving Flatness: Selecting the Honeywords from Existing User Passwords.

The system proposed in this paper works on issue to overcome the security problems. A new honeyword generation algorithm which shows better results with respect to flatness, DOS resistance and storage is proposed. New generation algorithm uses a different method that selects the honeywords from existing user passwords in the system which provides realistic honeywords and provides a new honeyword generation method and also reduces storage cost of the honeyword scheme.

3. HONEYWORDS

Basically, a simple but clever idea behind the study is the insertion of false passwords called as honeywords associated with each user’s account. When an adversary gets the password list, he recovers many password candidates for each account and he cannot be sure about which word is genuine. Hence, the cracked password files can be detected by the system administrator if a login attempt is done with a honeyword by the adversary. We use the notations and definitions depicted in Table 1 to simplify the description of the honeyword scheme.

TABLE 1. Notations

H()	Cryptographic hash function used to compute

	hash of the passwords
ui	Username for the ith user.
Pi	Password of the ith user
Xi	List of potential passwords for ui
Vi,j	Hash value of jth element of Xi
Vi	List of hash values for ui, Vi = (vi,1; vi,2; ...vi,k)
k	Number of elements in Xi
Ci	Index of the correct password in list Xi
Gene(k)	Procedure used to generate Xi of length k of sweetwords
sweetword:	Each element of Xi
sugarword:	Correct password in Xi
honeyword:	Each fake password of Xi

The honeyword mechanism works simply as follows: For each user ui, the sweetword list Xi is generated using the honey-word generation algorithm Gene(k). This procedure takes input k as the number of sweetwords and outputs both the password list Xi = (wi,1; wi,2; .... ; wi,k) and ci, where ci is the index of the correct password (sugarword). The username and the hashes of the sweetwords as < ui, (vi,1; vi,2; .... ; vi,k)> tuple is kept in the database of the main server, whereas ci is stored in another server called as honeychecker. By diversifying the secret information in the system storing password hashes in one server and ci in the honeychecker—makes it harder to compromise the system as a whole, i.e., providing a basic form of distributed security. Notice that in a traditional password technique < ui,H(pi) > pair is stored for each account, while for this system < ui, Vi > tuple is kept in the database, where Vi = (vi,1; vi,2; ..... ; vi,k). The login procedure of the scheme is summarized below:

- User ui enters a password g to login to the system. Server first checks whether or not H(g) is in list Vi.
- If not, then login is denied.
- Otherwise system checks to verify if it is a honeyword or the correct password.
- Let v (i, j) = H(g). Then j value is delivered to
- The honeychecker in an authenticated secure communication.
- The honeychecker checks whether j = ci or not.
- If the equality holds, it returns a TRUE value, otherwise it responses FALSE and may raise an alarm depending on security policy of the system
.

4. HONEYCHECKER

We assume that the system may incorporate an auxiliary secure server called the "honeychecker" to assist with the use of honeywords. Since we are assuming that the computer system is vulnerable to having the file F of password hashes stolen, one must also assume that salts and other hashing parameters can also be stolen. Thus, there is likely no place on the computer system where one can safely store additional secret information with which to defeat the adversary. The honeychecker is thus a separate hardened computer system where such secret information can be stored. We assume that the computer system can communicate with the honeychecker when a login attempt is made on the computer system, or when a user changes her password. We assume that this communication is over dedicated lines and/or encrypted and authenticated. The honeychecker should have extensive instrumentation to detect anomalies of various sorts. We also assume that the honeychecker is capable of raising an alarm when an irregularity is detected. The alarm signal may be sent to an administrator or other party different than the computer system itself. Depending on the policy chosen, the honeychecker may or may not reply to the computer system when a login is attempted. When it detects that something is amiss with the login attempt, it could signal to the computer system that login should be denied. On the other hand it may merely signal a "silent alarm" to an administrator, and let the login on the computer system proceed. In the latter case, we could perhaps call the honeychecker a "login monitor" rather than a "honeychecker." Our honeychecker maintains a single database value c(i) for each user ui; the values are small integers in the range 1 to k, for some small integer parameter k (e.g. k = 20). The honeychecker accepts commands of exactly two types:

• **Set: i, j**
Sets c(i) to have value j.

• Check: i, j
Checks that c(i) = j. May return result of check to requesting computer system . It may raise an alarm if check fails.

5. HONEYWORDS GENERATION METHODS

The authors categorize the honeyword generation methods into two groups. The first category consists of the legacy-UI (user interface) procedures and the second one includes modified-UI procedures whose password-change UI is modified to allow better password/honeyword generation. Take-a-tail method is given as an example of the second category. According to this approach a randomly selected tail is produced for the user to append this suffix to her entered password and the result becomes her new password. For instance, let a user enter password games01, and then system let propose '413' as a tail. So the password of the user now becomes games01413. Although this method strengthens the password, to our point of view, it is impractical—some users even forget the passwords that they determined. Therefore in the remaining parts, the analysis that we conducted is limited with the legacy-UI procedures. Note that some discussed points are indeed mentioned, but we emphasize those to address the paramount importance of the selected generator algorithm in terms of security.

5.1 Chaffing-by-Tweaking

In this method, the user password seeds the generator algorithm which tweaks selected character positions of the real password to produce the honeywords. For instance, each
character of a user password in predetermined positions is replaced by a randomly chosen character of the same type: digits are replaced by digits, letters by letters, and special characters by special characters. Number of positions to be tweaked, denoted as t should depend on system policy. As an example t = 3 and tweaking last t characters may be a method for the generator algorithm Gene(k, t). Another approach named in the study as "chaffing-by-tweaking-digits" is executed by tweaking the last t positions that contain digits. For example, by using the last technique for the password 52hungary and t = 2, the honeywords 10hungary and 65hungary may be generated

5.2 Chaffing-with-a-Password-Model

In this approach, the generator algorithm takes the password from the user and relying on a probabilistic model of real passwords it produces the honeywords. The authors give the model as an example for this method named as the modeling syntax. In this model, the password is splitted into character sets. For instance, mice3blind is decomposed as four-letters + one-digit + five-letters → L4 + D1 + L5 and replaced with the same composition like gold5rings.

Algorithm 1. SimpleModel algorithm

```
1: procedure SimpleModel (L)
2: w <---- random (L)       > randomly returns a word from L
3: d <---- length (w)        >returns length of word w
4: honeyword (1) <---- w (1)    > The first character is the just
                                  first character of w
5: for j <--- 2 to d do          >Probabilities of mod1, mod2 and
                              else are 0.1, 0.4 and 0.5
6: if mod1 then
7: w <---- random (L),  honeyword (j) <---- w (j)        >Add
              character in same position of new random word
8: else if mod2 then
9: w <---- random (L), honeyword (j) <---- w (j)       >Select
                a random word s.t. w(j - 1) =  honeyword(j - 1)
10: else
11: honeyword (j) <--- w (j)
                                     >Proceed with the same word
12: end if
13: end for
14: end procedure
```

5.3 Chaffing with "Tough Nuts"

In this method, the system intentionally injects some special honeywords, named as tough nuts, such that inverting hash values of those words is computationally infeasible, e.g. fixed length random bit strings should be set as the hash value of a honeyword. An illustrative example for a toughnut is given as '9,10PEe[KV.0?RUOtcL-:IJ"b+Wol !*]! NAT/pb'. It is stated that the number and positions of toughnuts are selected randomly. By means of this, it is expected that the adversary cannot seize whole sweetword set and some sweetwords will be blank for her, thereby deterring the adversary to realize her attack. It is discussed that in such a situation the adversary may pause before attempting login with cracked passwords.

5.4 Hybrid Method

Another method discussed is combining the strength of different honeyword generation methods, e.g. chaffing-with a- password-model and chaffing-by-tweaking-digits. By using this technique, random password model will yield seeds for tweaking digits to generate honeywords. For example let the correct password be orange2103. Then the honeywords apple2162 and saddy9138 should be produced as seeds to chaffing-by-tweaking-digits. For t = 3 and k = 4 for each seed, the sweetword table given below may be attained:

saddy9679	orange2422	apple2656
saddy9757	orange2903	apple2036
saddy9743	orange2172	apple2849
saddy9392	orange2792	apple2562

5.5 SECURITY ANALYSIS OF HONEYWORDS

In this part, we investigate the security of the honeyword system against some possible scenarios.

5.6 Denial-of-Service Attack

A denial-of-service (DoS) attack is discussed for the following scenario: Adversary knows the used Gene() procedure and can produce all possible honeywords for a given a password. For example, if the chaffing-by-tweaking-digits is employed in the system and with a small t adversary may generate whole possible honeywords from a known password. Consider the case, let password of a user be test42, then for t = 2 she can generate 100 possible honeywords and k of these honeywords are stored in the system password list. Let Pr ( ) = wij (i) denote the probability of correctly guessing a valid honeyword of Xi, where correct password pi is available to the adversary. Hence if this probability is a non negligible value, the adversary may attempt to login with the guessed honeyword to trigger an alarm condition. In fact, this may be serious, if a strong policy is set by the administrator e.g. a global password reset in response to a single honeyword hit. In the above example for k =20 and t =2, Pr(g = wi|pi) =(k - 1)/99 = 0:19. In order to mitigate this risk, the authors suggest choosing a relatively small set of honeywords randomly from a larger class of possible sweetwords. For the previous example, success probability of the attacker is about 19 percent for k =20, while this chance is decreased to 2 percent by only changing t = 3.

5.7 Brute-Force Attack

In the previous attack, we point out that if a strict policy is executed in a honeyword detection, system may be vulnerable to DoS attacks affecting the whole system. On the other hand, a soft policy weakens the influence of honeywords. In this regard, we describe the following attack to demonstrate an adversary can capture an amount of accounts in case of a light policy. We suppose an adversary has obtained a password file F and cracked numerous user passwords. Then, she tries to login with any accounts in the list instead of compromising a specific account. Furthermore, we assume that the adversary has no advantage in guessing the correct password by analyzing corresponding honeywords, i.e., Pr(g = pi) = 1/k. Last, if one of the user's honeywords is entered, the system takes the appropriate action according to one of the example policies as follows:

- Login proceeds as usual,
- User's account is shut down until the user establishes a new password.

The common point of the above policies is that even a honeyword entrance is detected, the system gives a local or no response. As a result of this, an adversary can carry out a brute-force search until a successful login is obtained. For example, even a user's account is locked due to a honeyword attempt, she continues to search with another user's account, i.e., single guess for each user. She likely makes a correct guess after k trials, since Pr(g =pi) = 1/k. As an illustrative example for k = 20, it is highly possible that the adversary finds a correct password after 20 attempts.

6. CONCLUSIONS

In this study, we have analyzed the security of the honeyword system and addressed a number of flaws and issues that need to be handled before successful realization of the scheme. Also our system helps to identify that the attack has been taking place due to use of Honeychecker. In this respect, we have pointed out that the strength of the honeyword system directly depends on the generation algorithm, i.e., flatness of the generator algorithm determines the chance of distinguishing the correct password out of respective sweetwords. Furthermore, we have demonstrated the weak and strong points of each method introduced in the original study. In the future, we would like to refine our model by involving hybrid generation algorithms to also make the total hash inversion process harder for an adversary in getting the passwords in plaintext form from a leaked password hash file. Hence, by developing such methods both of two security objectives increasing the total effort in recovering plaintext passwords from the hashed lists and detecting the password disclosure can be provided at the same time.

REFERENCES

[1] D. Mirante and C. Justin, “Understanding password database compromises,” Dept. of Computer Science Polytechnic Inst. of NYU, New York, NY, USA: Tech. Rep. TR-CSE-2013-02, 2013.

[2] A. Juels and R. L. Rivest, “Honeywords: Making password cracking detectable,” in Proc. ACM SIGSAC Conf. Computer. Commun. Security, 2013, pp. 145–160.

[3] K. Brown, “The dangers of weak hashes,” SANS Institute InfoSec Reading Room, Maryland US, pp. 1–22, Nov. 2013, http://www.sans.org/reading-room/ whitepapers/authentication/dangers-weak-hashes-34412.

[4] A. Vance, “If your password is 123456, just make it hack me,” New York Times, Jan. 2010.

[5] C. Herley and D. Florencio, “Protecting financial institutions from brute-force attacks,” in Proc. 23rd Int. Inform. Security Conf., 2008, pp. 681–685.

[6] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, “Kamouflage: Loss-resistant password management,” in Proc. 15th Eur. Conf. Res. Comput. Security, 2010, pp. 286–302.

[7] J. A and L. R. R, "Honeywords: Making Password cracking Detectable," in ACM SIGSAC conference on Computer & communications security, November 2013.

[8] Imran Erguler, "Achieving Flatness: Selecting the Honeywords from Existing User Passwords," IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 2, pp. 284 - 295, February 2015.

[9] M. Dell'Amico, P. Michiardi and Y. Roudier, "Password Strength: An Empirical Analysis," INFOCOM'10: Proceedings of the 29th Conference on Information Communications, vol. 10, pp. 983-991, 2010.

[10] Mirante, Dennis and Justin Cappos, "Understanding Password Database Compromises," Dept. of Computer Science and Engineering Polytechnic Inst. of NYU, 2013.

[11] R. Morris and K. Thompson, "Password Security: A Case History," Communications of the ACM, vol. 22, no. 11, pp. 594-597, 1979.

[12] 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. In IEEE Symposium on Security and Privacy (SP), pages 523–537, 2012.

[13] C. Herley and P. Van Oorschot. A research agenda acknowledging the persistence of passwords. IEEE Security & Privacy, 10(1):28–36, 2012.