Detection & Identification of Attacker Using Honey Words In A Purchase Portal

S.Arthi[1], D.Siva[2]
Department of Computer Science and Engineering
[1]M.E Student, IIET Anna University Chennai
[2]Assistant Professor, IIET Anna University Chennai

Abstract-*Recently, Juels and Rivest proposed honeywords (decoy passwords) to detect attacks against hashed password databases. For each user account, the legitimate password is stored with several honeywords in order to sense impersonation. If honeywords are selected properly, a cyber-attacker who steals a file of hashed passwords cannot be sure if it is the real password or a honeyword for any account. Moreover, entering with a honeyword to login will trigger an alarm notifying the administrator about a password file breach. In this study, we scrutinize the honeyword system and present some remarks to highlight possible weak points. Also, we suggest an alternative approach that selects the honeywords from existing user passwords in the system in order to provide realistic honeywords—a perfectly flat honeyword generation method—and also to reduce storage cost of the honeyword scheme.*

Keywords- Authentication, honeypot, honeywords, login, passwords, password cracking

I. INTRODUCTION

Disclosure of password files is a severe security prob-lem that has affected millions of users and companies like Yahoo, RockYou, LinkedIn, eHarmony and Adobe [1],

[2], since leaked passwords make the users target of many possible cyber-attacks. These recent events have demon-strated that the weak password storage methods are cur-rently in place on many web sites. For example, the LinkedIn passwords were using the SHA-1 algorithm with-out a salt and similarly the passwords in the eHarmony sys-tem were also stored using unsalted MD5 hashes [3]. Indeed, once a password file is stolen, by using the pass-word cracking techniques like the algorithm of Weir et al. [4] it is easy to capture most of the plaintext passwords.

In this respect, there are two issues that should be con-sidered to overcome these security problems: First, passwords must be protected by taking appropriate pre-cautions and storing with their hash values computed through salting or some other complex mechanisms. Hence, for an adversary it must be hard to invert hashes to acquire plaintext passwords.

The second point is that a secure system should detect whether a password file dis-closure incident happened or not to take appropriate actions. Honeypot is one of the methods to identify occurrence of a password database breach. In this approach, the admin-istrator purposely creates deceit user accounts to lure adversaries and detects a password disclosure, if any one of the honeypot passwords get used [5], [6]. This idea has been modified by Herley and Florencio [7] to protect online banking accounts from password brute-force attacks. According to the study, for each user incorrect login attempts with some passwords lead to honeypot accounts, i.e., malicious behavior is recognized. For instance, there are 108 possibilities for a eight-digit pass-word and let system links 10,000 wrong password to hon-eypot accounts, so the adversary performing the brute-force attack 10,000 times more likely to hit a honeypot account than the genuine account.

Use of decoys for building theft-resistant was introduced by Bojinov et al. in [8] called as Kamouflage. In this model, the fake pass-word sets are stored with the real user password set to conceal the real passwords, thereby forcing an adversary to carry out a considerable amount of online work before getting the correct information. Recently, Juels and Rivest have presented the honeyword mechanism to detect an adversary who attempts to login with cracked passwords [9]. Basically, for each username a set of sweetwords is constructed such that only one element is the correct password and the others are honeywords (decoy pass-words). Hence, when an adversary tries to enter into the system with a honeyword, an alarm is triggered to notify the administrator about a password leakage. The details of the method will be given in the next section. In this study, we analyze the honeyword approach and give some remarks about the security of the system. The rest of this paper is organized as follows. In Section 2, we review the honeyword approach and discuss the honeyword generation procedures. Section 3 examines security of these procedures and Section 4 gives the description of our proposed model. In Section 5, we ana-lyze its security properties and demonstrate a comparison between

our approach and the original methods in Section 6. Finally, in Section 7 we conclude this paper.

II. HONEYWORDS

In this section, we first briefly summarize the honeyword password model proposed by Juels and Rivest in [9]. Then, we overview the methods on generation of honeywords given in the study and discuss some points that can cause some security problems.

Review of Honeywords

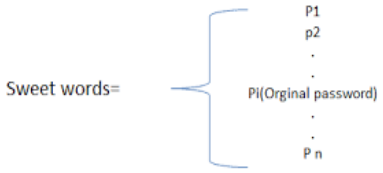
Basically, a simple but clever idea behind the study is the insertion of false passwords—called as honeywords—associated with each user's account. When an adversary gets the password list, she recovers many password candi-dates for each account and she cannot be sure about which word is genuine.

Hence, the cracked password files can be detected by the system administrator if a login attempt is done with a honeyword by the adversary.

Honeyword Generation

Methods and Discussions

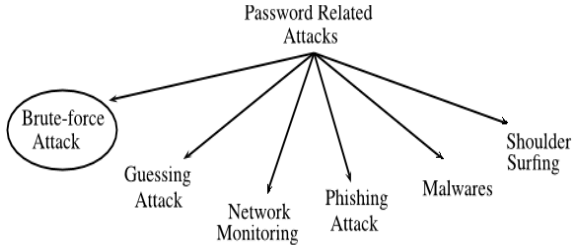
The authors in [9] categorize the honeyword generation methods into two groups. The first category consists of the legacy-UI (user interface) procedures and the second one includes modified-UI procedures whose password-change UI is modified to allow better password/honeyword gen-eration. Take-a-tail 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.

III. SECURITY ANALYSIS OF HONEYWORDS

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



3.1 Denial-of-Service Attack

In [9], a denial-of-service (DoS) attack is discussed for the following scenario: Adversary knows the used GenðÞ proce-dure 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 pass-word.

3.2 Brute-Force Attack

In the previous attack, we point out that if a strict policy is executed in a honeyword detection, system may be vulnera-ble 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 ass-ume that the adversary has no advantage in guessing the correct password by analyzing corresponding honey-words, i.e., Prðg ¼ p_iÞ ¼ 1=k. Last, if one of the user's hon-eywords 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 estab-recognizes that each username is paired with k numberslishes a new password.

3.3 Choosing Policy

By considering the described attacks and discussions, one can infer that there are two major issues about honeywords. The first issue is flatness of the generator

algorithm such that it is directly related to the chance of distinguishing the correct password out of the respective sweetwords. Thus, if the method is not flat enough, it undermines the main task of the honeywords and an adversary can easily perceive the correct password. Second issue is that what is the chance of an adversary in hitting a honeyword intentionally and trig-gering a false alarm to render the system in a DoS state. Sig-nificance of this issue depends on the adapted policy.Also a limit, as _, for the maximum number of honeyword attempts in a period should be set to prevent the brute-force attack.

IV. A NEW APPROACH

Our proposed model is still based on use of honeywords to detect password-cracking. However, instead of gener-ating the honeywords and storing them in the password file, we suggest to benefit from existing passwords to sim-ulate honeywords. In order to achieve this, for each account k _ 1 existing password indexes, which we call honeyindexes, are randomly assigned to a newly created account of ui, where k _ 2. Moreover, a random index number is given to this account and hash of the correct password is kept with the correct index in a list. On the other hand, in another list ui is stored with an integer set which is consisted of the honeyindexes and the correct index.

TABLE 2
Example Password File F_1 for the Proposed Model

Username	Honeyindex Set
agent-lisa	ð93; 16; 626; . . . ; 94; 931Þ
Alexius	ð15; 476; 51; 443; . . . ; 88; 429Þ
baba13	ð3; 62107; . . . ; 91; 233Þ
:	:
zack_tayland	ð1; 009; 23; 471; . . . ; 47; 623Þ
zoom42	ð63; 51234; . . . ; 72; 382Þ

Initialization

First, T fake user accounts (honeypots) are created with their passwords (see Appendix A for details). Also an index value between ½1; N&, but not used previously is assigned to each honeypot randomly. Then k _ 1 numbers are randomly selected from the index list and for each account a honeyindex set is built like Xi ¼ ðxi;1; xi;2; . . . ; xi;kÞ; one of the elements in Xi is the correct index (sugarindex) as ci. Now, we use two password files as F1 and F2 in the main server: F1 stores username and honeyindex set, <hui; Xi> pairs as shown in Table 2, where hui denotes a honeypot account. Note that each entry has two elements. The first one is the username of the account and the second element is honeyindex set for the respective account. Also, the table is sorted alphabetically by the username field.

On the other hand, F2 keeps the index number and the corre-sponding hash of the password, < ci; HðpiÞ >, as depicted in Table 3. In this case, each entry in the table has two ele-ments. The first element is the sugarindex of the account and the second one is the hash of the corresponding pass-word. Notice that the table is sorted according to the index values. Let SI denote the index column and SH represent the corresponding password hash column of F2. Then the function fðciÞ that gives password hash value in SH for the index value ci can be defined as: fðciÞ ¼ fHðpiÞ 2 SH : < ci; HðpiÞ > stored pair of ui and ci 2 SI g. In order to make points clear, the initialization process is shown within the following example.

Example 1. Suppose that a honeypot username/password pair is generated like < macbeth; master2014 > by the system. Then an index number is randomly selected, for instance 1,008, and assigned as the correct index of this account. Now F2 file is updated according to this infor-mation as shown below:

Index No	Hash of Password
:	:
1,008	Hðmaster2014Þ
:	:

TABLE 3
Example Password File F_2 for the Proposed Model

S_I	S_H
3	$Hðp_3Þ$
7	$Hðp_7Þ$
85	$Hðp_{85}Þ$
:	:
100,000	$^{H}ðp^{p}10000 0Þ$
100,004	$^{H}ðp^{p}100004 Þ$

Then, k _ 1 numbers are randomly chosen from SI of F2 and combined with correct index 1008 in a random manner to produce the index group. For instance, if k ¼ 5, such a group ð42; 96; 104; 1;008; 7;201; 23;008Þ may be generated. In this case F1 file is seen as below:

Username	Honeyindex Set
:	:
macbeth	ð42; 96; 104; 1;008; 7;201; 23;008Þ
:	:

4.2 Registration

After the initialization process, system is ready for user reg-istration. In this phase, a legacy-UI is preferred, i.e., a user-name and password are required from the user as u_i; p_i to register the system. We use the honeyindex generator algo-rithm Genðk; S_I Þ ! c_i; X_i, which outputs c_i as the correct index for u_i and the honeyindexes X_i ¼ ð$x_{i;1}$; $x_{i;2}$; . . . ; $x_{i;k}$Þ. Note that Genðk; S_I Þ produces X_i by randomly selecting k _ 1 numbers from S_I and also randomly picking a number c_i 2= S_I . So c_i becomes one of the elements of X_i. One can see that the generator algorithm Genðk; S_I Þ is different from the procedure described in [9], since it outputs an array of inte-gers rather than a group of honeywords.

4.3 Honeychecker

In our approach, the auxiliary service honeychecker is employed to store correct indexes for each account and we assume that it communicates with the main server through a secure channel in an authenticated manner. Indeed, it can be assumed that security enhancements for honeychecker and the main server presented in [16] are applied, but it is out scope of this study.

The role and primary processes of the honeychecker are the same as described in the original study [9], except that < i; c_i> pair is replaced with < u_i; c_i> pair in our case. The honeychecker executes two commands sent by the main server:

- Set: c_i,u_i
- Sets correct password index c_i for the user u_i. Check: u_i; j

Checks whether c_i for u_i is equal to given j. Returns the result and if equality does not hold, notifies system a honeyword situation.

Thus, the honeychecker only knows the correct index for a user-name, but not the password or hash of the password.

4.4 Login Process

System first checks whether entered password, g, is correct for the corresponding username u_i. To accomplish this, first the X_i of the corresponding u_i is attained from the F_1 file. Then, the hash values stored in F_2 file for the respective indices in X_i are compared with HðgÞ to find a match. If a match is not obtained, then it means that g is neither the cor-rect password, nor one of the honeywords, i.e., login fails. On the other hand, if HðgÞ is found in the list, then the main server checks whether the account is a honeypot. If it is a honeypot, then it follows a predefined security policy against the password disclosure scenario. Notice that for a honeypot account there is no importance of the entered password is genuine or a honeyword, so it directly manages the event without communicating with the honeychecker. If, however, HðgÞ is in the list and it is not a honeypot, the corresponding j 2 X_i is delivered to honeychecker with username as < u_i; j > to verify it is the correct index. Hon-eychecker controls whether j ¼ c_i and returns the result to the main server. At the same time, if it is not equal, then it assured that the proffered password is a honeyword and adequate actions should be taken depending on the policy.

V. SECURITY ANALYSIS OF THE PROPOSED MODEL

In this section, we investigate the security of the proposed model against some possible attack scenarios. Before,however, we elaborate on the attack strategies, we will first state a set of reasonable assumptions about our approach and the related security policies. We suppose that the adver-sary can invert most or many of the password hashes in file F_2. Notice that the introduction of this scheme comes with a DoS attack sensitivity in which an adversary deliberately tries to login with honeywords to trigger a false alarm. Hence, the suggested policies given below mostly focuses on minimizing the DoS vulnerabilities.

5.1 DoS Attack

Under this attack scenario as described in Section 3.1, the adversary does not have the password files and their con-tents. Her main purpose is to trigger a false alarm and to raise a honeyword alarm situation, i.e., depending on the policy some or all parts of the system may be out of service or disabled unnecessarily. We suppose that the adversary has knowledge m þ 1 username and respective passwords in the system as ðu_a; p_a; . . . ; $u_{aþm}$; $p_{aþm}$ Þ; maybe she inten-tionally created all of these accounts. In this case, a plausible method for attacking the system is creating m accounts with the same password as p_z, while a single account, u_y, has dif-ferent password like p_y and entering the system with the username u_y and the password p_z. If p_z is assigned by the system as a honeyword, then the adversary mounts a DoS attack by entering with the system < u_y; p_z> pair. Let Prðp_z 2 W_yÞ denote the probability that p_z is assigned as one of the honeywords for u_y; it is also the success probability of the adversary for this attack. Since there are N _ m pass-words different from p_z[1] and k honeywords are assigned to each account:

$$\Pr p_z \quad {}^{W}y \qquad 1 \quad \_N \_ m^{-k} : \qquad (1)$$
$$\eth \ 2 \qquad \text{Þ}¼ \quad \_ \qquad N$$

As an illustrative example for N ¼ 1;000;000, k ¼ 20 and m ¼ 100, from Eq. (1) an adversary succeeds in realizing the described attack with a probability of 0:002. Note that, the success of the adversary directly depends on ðm=NÞ, so for large values the chance of the adversary will be increased. For instance if N ¼ 1;000, m ¼ 10 and k ¼ 20 (as an extreme example, one out of 100 accounts is created by the adversary), the success probability of the adversary will be 0:18.

5.2 Password Guessing

In this attack, we assume that the adversary has plundered password files F_1 and F_2 from the main server and also obtained plaintext passwords by inverting the hash values. Extracted F_2 file (after inverting hashes) gives < indexnumber; password > pairs to the adversary, but they are not directly connected to a specific username. By just analyzing this, she cannot exactly determine which pass-word belongs to which user. On the other hand, F_1 gives username; indexset pairs such that for each username k pos-sible passwords exist.

5.3 Brute-Force Attack

In this part, we consider the attack described in Section 3.2. We suppose that if a honeypot entrance is detected by the system, it responds with a strong reaction, while a light pol-icy (not suggested) is executed in case of a honeyword detection. So, we assume that even in a honeyword detec-tion the adversary may proceed to make her trials due to light local policies. If, however, a honeypot account is attempted then system follows a strong policy.

VI. COMPARISON OF HONEYWORD GENERATION MODELS

In this section, we give a comparison of the generation methods including our proposed model with respect to storage cost, DoS resistance and flatness of each algorithm. Before discussing these issues in detail, we would like to talk about how the proposed model changes total hash inversion effort of an adversary who has a leaked password file (F_1 and F_2 files for our case). In fact, as mentioned in Section 1, defending and detecting are two different issues from the point of password security. For example, by realizing the salted-high iteration password storage techniques, inverting a hash from a captured password file becomes time consuming.

VII. CONCLUSION

In this study, we have analyzed the security of the honey-word system and addressed a number of flaws that need to be handled before successful realization of the scheme. In this respect, we have pointed out that the strength of the honeyword system directly depends on the genera-tion algorithm, i.e., flatness of the generator algorithm determines the chance of distinguishing the correct pass-word out of respective sweetwords. Another point that we would like to stress is that defined reaction policies in case of a honeyword entrance can be exploited by an adversary to realize a DoS attack. This will be a serious threat if the chance of an adversary in hitting a honey-word given the respective password is not negligible.

REFERENCES

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

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

[3] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, “Password cracking using probabilistic context-free grammars,” in Proc. 30th IEEE Symp. Security Privacy, 2009, pp. 391–405.

[4] F. Cohen, “The use of deception techniques: Honeypots and decoys,” Handbook Inform. Security, vol. 3, pp. 646–655, 2006.

[5] M. H. Almeshekah, E. H. Spafford, and M. J. Atallah, “Improving security using deception,” Center for Education and Research Information Assurance and Security, Purdue Univ., West Lafayette, IN, USA: Tech. Rep. CERIAS Tech. Rep. 2013-13, 2013.

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

[8] M. Burnett. The pathetic reality of adobe password hints. [Online]. Available: https://xato.net/windows-security/adobe-password-hints, 2013.

[9] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford, “CAPTCHA: Using hard ai problems for security,” in

Proc. 22nd Int. Conf. The-ory Appl. Cryptographic Tech., vol. 2656, 2003, pp. 294–311.

[10] L. Zhao and M. Mannan, "Explicit authentication response con-sidered harmful," in Proc. Workshop New Security Paradigms Work-shop, 2013, pp. 77–86.

[11] Z. A. Genc, S. Kardas, and M. S. Kiraz, "Examination of a new defense mechanism: Honeywords," IACR Cryptology ePrint Archive, Report 2013/696, 2013.