HONEYWORDS

Mr. Jayakumar.S[1], Shivraj[2], Pranav Dwivedi[3]

[1] *Assistant Professor (S.G), Dept. of Computer Science and engineering, SRM IST, Tamil Nadu, India*
[2] *Student Dept. of Computer Science and engineering, SRM IST, Tamil Nadu, India*
[3] *Student Dept. of Computer Science and engineering, SRM IST, Tamil Nadu, India*

ABSTRACT

It has become much easier to crack a password hash with the advancements in the graphical processing unit (GPU) technology. An adversary can recover a user's password using brute-force attack on password hash. Once the password has been recovered no server can detect any illegitimate user authentication (if there is no extra mechanism used). In this context, recently, Juels and Rivest published a paper for improving the security of hashed passwords. Roughly speaking, they propose an approach for user authentication, in which some false passwords, i.e., "honey words" are added into a password file, in order to detect impersonation. Their solution includes an auxiliary secure server called "honey checker" which can distinguish a user's real password among her honey words and immediately sets off an alarm whenever a honey word is used. In this paper, we analyze the security of the proposal and provide some possible improvements which are easy to implement.

Keyword : - *Authentication, Honeywords, login, passwords, Honeychecker.*

---

1. INTRODUCTION

Passwords are a notoriously weak authentication mechanism. Users frequently choose poor passwords. 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. 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. It becomes much easier to crack a password hash. An adversary can recover a user's password using brute-force attack on password hash. Once the password has been recovered no server can Detect any illegitimate user authentication. So Honeywords plays an important role to defense against stolen password files. Honeywords enables detection of theft, prevents impersonation Honeywords are ``decoy passwords'' (many for each user) Separate ``honeychecker'' aids in password checking .

1.1 Aim & Objective

Monitoring data access patterns where system will generate honeywords to keep user data secure. Decoy data will be stored in the database, alongside the users real data also serve as sensors to detect illegitimate access or exposure is suspected. To validate the alerts issued by the anomaly detector that monitors user access behavior. Launch a disinformation attack by returning large amounts of decoy information to the attacker.

1.2 Problem Definition

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. At the expense of increasing the storage requirement by 20 times, the authors introduce a simple and effective solution to the detection of password file disclosure events. 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.

1.3 Motivation

Real passwords are often weak and easily guessed; either by sharing passwords, using names of loved ones, dictionary words, and brute force attacks. Motivation towards this project is to prevent the attacks and keep the adversaries away from the user accounts. Theft of password hash files are increasing. Therefore, this technique will give a break to hackers. Adversary compromises systems, steal password hashes, and cracks the hash. Adversary makes changes in the hash files, or misuse with the user accounts, eaves dropping and many more. Adversary succeeds in impersonating legitimate user and login.

2. TECHNICAL DESCRIPTION

We assume a computer system with n users u1, u2, . . . , un; here ui is the username for the ith user. We let pi denote the password for user ui. This is the correct, legitimate, password; it is what user ui uses to login to the system. In current practice, the system uses a cryptographic hash function H and stores hashes of passwords rather than raw passwords. That is, the system maintains a file F listing username / password-hash pairs of the form (ui, H(pi)) for i = 1, 2, . . . ,n.

2.1 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.

2.2 Honeyword generation

The procedures split according to whether there is an impact on the user interface (UI) for password change. (The login procedure is always unchanged.) We distinguish the two cases: With legacy- UI procedures, the password-change UI is unchanged. This is arguably the more important case. We propose two legacy -UI procedures: chaffing-by- tweaking (which includes chaffing-by-tail-tweaking and chaffing-by-tweaking-digits as special cases), and chaffing-with-a-password-model.With modified UI procedures, the password-change UI is modified to allow for better password/ honeyword generation. We propose a modified -UI procedure called take-a-tail. With take-a-tail, the UI change is really very simple: the user's new password is modified to end with a given, randomly-chosen three-digit value. Otherwise take-a-tail is the same as chaffing-by-tail-tweaking. Chaffing: The password pi is picked, and then the honeyword generation procedure gen(k; pi) or "chaff procedure" generates a set of $k - 1$ additional distinct honeywords ("chaff"). Note that the honeywords may depend upon the password pi. The password and the honeywords are placed into a list Wi, in random order. The value c(i) is set equal to the index of pi in this list. The success of chaffing depends on the quality of the chaff generator; the method fails if an adversary can easily distinguish the password from the honeywords.

2.2.1 Chaffing by tweaking

Our first method is to “tweak” selected character positions of the password to obtain the honeywords. Let t denote the desired number of positions to tweak (such as t = 2 or t = 3). Using this honeyword generation method honeywords are randomly generated.

```
Take input as a position (pos) and password (pass).
Apply for loop from 1 to 10.
if (i==pos)
realPassword[i] =pass;
hashPassword[i] =generateHash(pass);
else
realPassword[i] =replace(pass1);
hashPassword[i] =generateHash(pass);
PassResult.put("Real",realPassword);
PassResult.put("Hash",HashedPassword);
returnPassResult;
```

2.2.2 Chaffing-with-a-password-model

Our second method generates honeywords using a probabilistic model of real passwords; this model might be based on a given list L of thousands or millions of passwords and perhaps some other parameters. (Note that generating honeywords solely from a published list L as honeywords is not in general a good idea: such a list may also be available to the adversary, who could use it to help identify honeywords.) Unlike the previous chaffing methods, this method does not necessarily need the password in order to generate the honeywords, and it can generate honeywords of widely varying strength.

3. ARCHITECTURE

3.1 System Architecture Diagram

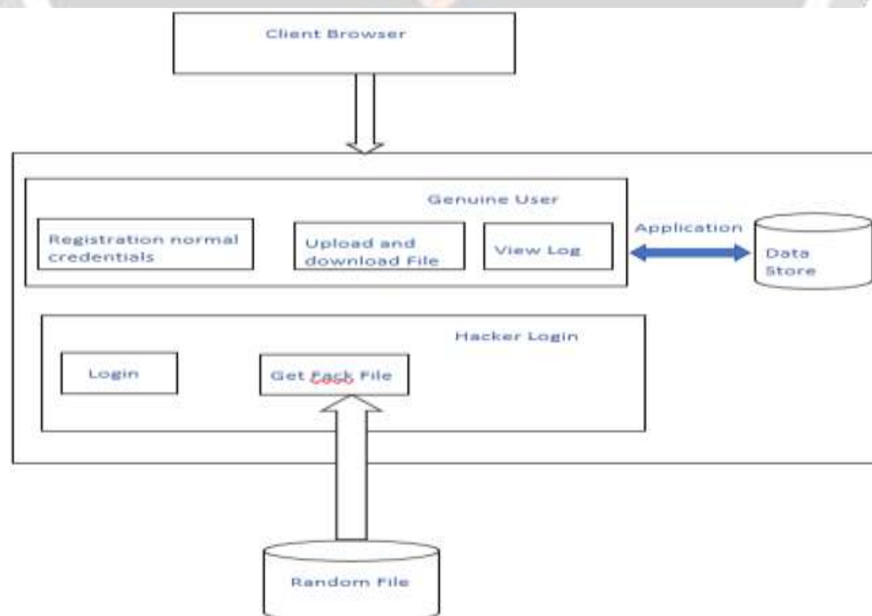

Fig -1: System Architecture Diagram

3.2 Dataflow diagram

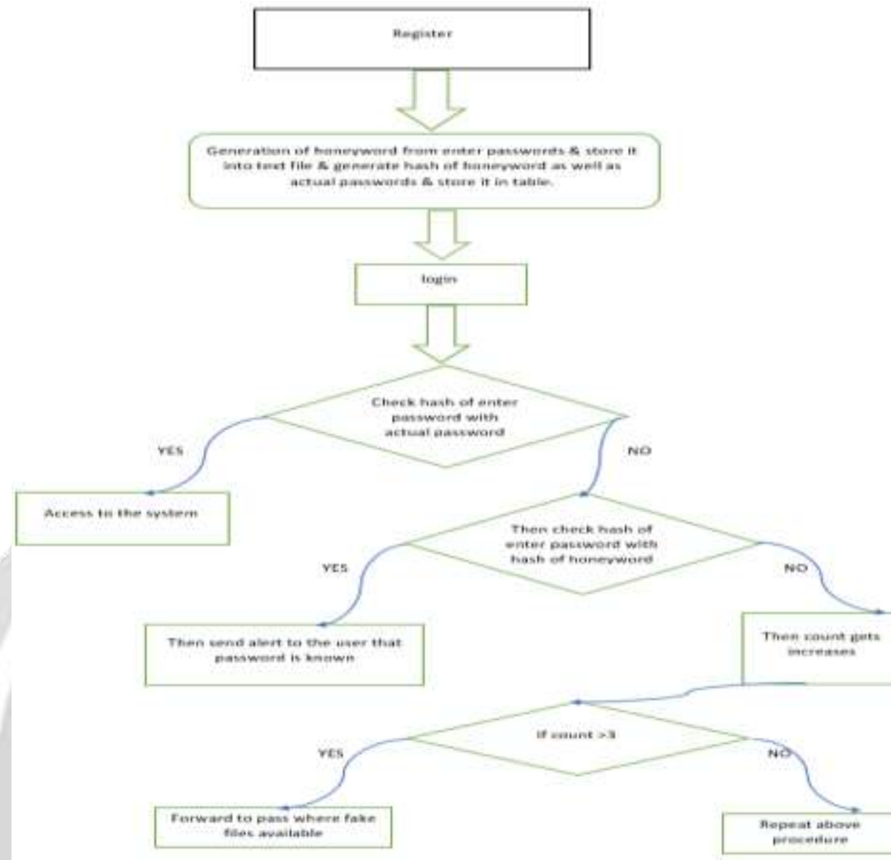

Fig -2:Dataflow diagram

4. MODULES

Our proposed model is still based on use of honeywords to detect password-cracking. However, instead of generating honeywords and storing them in the password _le, we suggest to bene_t from existing passwords to simulate honey-words. 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 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 hon- eyindexes and the correct index. So, when an adversary analyzes the two lists, she recognizes that each username is paired with k numbers as sweetindexes and each of which points to real passwords in the system. The tentative password indexes hampers an adversary to make a correct guess and she cannot be easily sure about which index is the correct one. It is equivalent to say that to create uncertainty about the correct password, we propose to use indexes that map to valid passwords in the system. The contribution of our approach is twofold. First, this method requires less storage compared to the original study. Second, in previous sections we argue that e_ectiveness of the honeyword system directly depends on how Gen() atness is provided and how it is close to human behaviour in choosing passwords. Within our approach passwords of other users are used as fake passwords, so guess of which password is fake and which is correct becomes more complicated for an adversary.

4.1 Initialization

Firstly, T fake user accounts (honeypots) are created with their passwords. Also an index value between [1;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 honey index set is built like Xi = (xi;1; xi;2; : : : ; xi;k); one of the elements in Xi is the correct index (sugarindex) as ci. Now, we use two password _les as F1 and F2 in the main server: F1 stores username and honeyin- dex set, < hui;Xi > pairs as shown in Table 2, where hui denotes a honeypot accounts. On the other hand F2 keeps index number and corresponding hash of password, < ci;H(pi) >, as depicted in Table 3. Let SI denote index column and SH represent the corresponding password hash column of F2. Then the function f(ci) that gives password hash value in SH for the index value ci can be de_ned as: f(ci) = fH(pi) 2 SH :< ci;H(pi) > stored pair of ui and ci 2 SIg.

4.2 Registration

After the initialization process, system is ready for user registration. In this phase, a legacy-UI is preferred, i.e. a username and password are required from the user as ui; pi to register the system. We use the honeyindex generator algo-rithm Gen(k; SI) ! ci;Xi, which outputs ci as the correct index for ui and the honeyindexes Xi = (xi;1; xi;2; : : : ; xi;k). Note that Gen(k; SI) produces Xi by randomly selecting k 1 numbers from SI and also randomly picking a number ci =2 SI . So ci becomes one of the elements of Xi. One can see that the generator algorithm Gen(k; SI) is di_erent from the procedure described in [9], since it outputs an array of integers rather than a group of honeywords. Note, however, that the index array Xi is indeed represents which honeywords are assigned for ui.

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; ci > pair is replaced with < ui; ci > pair in our case. The honeychecker executes two commands sent by the main server. The honeychecker only knows the correct index for a username, but not the password or hash of the password.

4.4 Login Process

System firstly checks whether entered password, g, is correct for the correspond- ing username ui. To do this, the hash values stored in F2 _le for the respective indices in Xi are compared with H(g) to _nd a match. If a match is not obtained, then it means that g is neither the correct password nor one of the honeywords, i.e. login fails. On the other hand, if H(g) 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 prede_ned 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, H(g) is in the list and it is not a honeypot, the corresponding j 2 Xi is delivered to honeychecker with username as < ui; j > to verify it is the correct index. Honeychecker controls whether j = ci and returns 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.

5.CONCLUSIONS

We present an approach where business data and personal data can be secured. We propose a monitoring of a person or intruder, also its illegal actions performed on any system are seen. Decoy documents stored in a system containing user’s real data serve as sensors to detect an unauthorised access. Once the data gathered is suspected and later verified, we can block the malicious insider with fake information in order to dilute the user’s real data. Such methods can be used to reduce the number of hacks and also provide security at social level. In future we would like to refine our model by involving hybrid hash algorithm to make the inversion process more harder, where the chances of attackers to hack or attack on any system gets reduced. The admin keeps the data of the tracked IP’s with them and use them to block access on their network. Use of honeywords is very useful and can be used by any user

account .By developing such methods two objectives can be maintained-Total efforts for getting plaintext passwords and hashed list and detecting password disclosure.

6. REFERENCES

[1] Prof S.P. Khedhar, Bhavana Bachhav , Pratiksha Parsewar, Rakshanda Tirmal "Achieving Flatness by Selecting the Honey words from Existing User Passwords".
[2] Juels and Rivest "Honeywords: Making Password-Cracking Detectable".
[3] Generating Honeywords from Real Passwords with Decoy Mechanism.
[4] M. Bakker and R. van der Jagt. GPU-based password cracking. Technical report, Univ. Of Amsterdam. [5] Brown, K.: The dangers of weak hashes. Technical report, SANS Institute InfoSec Reading Room (2013)