Examination of a New Defense Mechanism: Honeywords

Ziya Genc*, Süleyman Kardaş*,†, Mehmet Sabir Kiraz*

*TÜBİTAK BİLGEM UEKAE, Gebze, Kocaeli, Turkey
†Sabancı University, Faculty of Engineering and Natural Sciences, İstanbul, Turkey

***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., "honeywords" are added into a password file, in order to detect impersonation. Their solution includes an auxiliary secure server called "honeychecker" which can distinguish a user's real password among her honeywords and immediately sets off an alarm whenever a honeyword is used. In this paper, we analyze the security of the proposal and provide some possible improvements which are easy to implement.

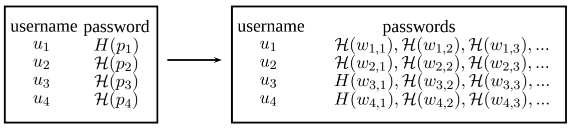
***Keywords*-Security, Authentication, Password, Honeywords**

I. Introduction

The use of passwords is one of the most common tools during authentication process. In registration process, most of the users chooses weak passwords that can be predicted by a brute-force attack. Namely, an adversary, who steal the file of hashed passwords from a server, can use brute force attack to recover some user's password. Weir et al. [16] developed a password cracking algorithm which uses probabilistic, context-free grammars. Kelley et al. [10] recently showed that using Weir's attack, one billion guess is enough to crack %40.3 of the passwords that comply with the "basic8" policy, i.e., all passwords must have at least 8 characters. Golubev showed that the cracking speed of hashes has reached 5.6 billion/s for MD5 and 2.3 billion/s for SHA1 on a single GPU [2]. These advancements make it necessary to develop new security measures.

In [9], Juels and Rivest recently propose the idea of changing the structure of the password file in such a way that each user would have multiple possible passwords, *sweetwords* and only one of them is real. The false passwords are called *honeywords*. As soon as one of the honeywords is submitted in the login process, the adversary will be detected. The idea works as follows.

Let u_i, p_i and $\mathcal{H}()$ denotes the i^{th} user name, her password and the hash function of the standard system respectively. As in Figure 1, the system adds honeywords hashes to this file at random orders. Thus an adversary who has recovered the password from the hash values will see randomly ordered sweetwords $w_{i,j}$ of user u_i.



username	password
u_1	$H(p_1)$
u_2	$\mathcal{H}(p_2)$
u_3	$\mathcal{H}(p_3)$
u_4	$\mathcal{H}(p_4)$

$\longrightarrow$

username	passwords
u_1	$\mathcal{H}(w_{1,1}), \mathcal{H}(w_{1,2}), \mathcal{H}(w_{1,3}), ...$
u_2	$\mathcal{H}(w_{2,1}), \mathcal{H}(w_{2,2}), \mathcal{H}(w_{2,3}), ...$
u_3	$H(w_{3,1}), \mathcal{H}(w_{3,2}), \mathcal{H}(w_{3,3}), ...$
u_4	$H(w_{4,1}), \mathcal{H}(w_{4,2}), \mathcal{H}(w_{4,3}), ...$

Fig. 1. The structure of the password hashes file of a standard system is on the left. The system using honeywords is on the right.

When a user u_i sends a login request, the server will determine the order of her among the users, and the order of the submitted password among her sweetwords. The server sends a message of the form $Check(i, j)$ to a secure server which is called "honeychecker", for the i^{th} user and her j^{th} sweetword. The honeychecker will determine whether the submitted word is a password or a honeyword. If a honeyword is submitted, then it will raise an alarm or take an action that is previously chosen (see Figure 2). The honeychecker does not know anything about the user's password or honeywords. It maintains a single database that contains only the index of the true password among the user's sweetwords.

The adversary can steal the file of hashed passwords from the server and invert the hashes but cannot tell which sweetword is the password. There will be a risk of detection that prevents the adversary to successfully

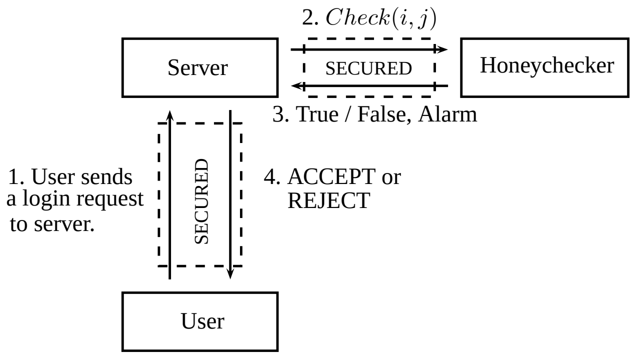


Fig. 2. Login schema of a system using honeywords

access to the system.

A closely related work is the Kamouflage system of Bojinov et al. [3] but this work differs from honeywords. In this system, the user's password list is placed with other lists, each of which contains different honeywords. In Kamouflage system, there is no need to use a server as honeychecker. Nevertheless, Bojinov et al.claim that the use of the server can increase the ability of detection of compromise [3].

Our contribution. In this work, we analyze the honeyword system according to both functionality and the security perspective. Then, we suggest three practical improvements; number of honeywords per user, generating typo-safe honeywords and managing old passwords.

II. Attack Models

There are numerous attacks to obtain a user's password. The six of these techniques are depicted in Figure 3.

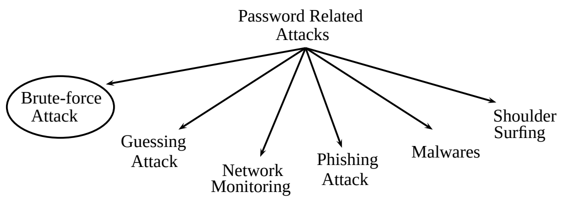
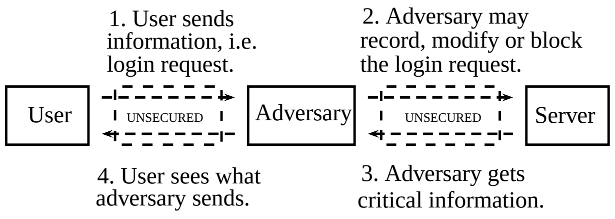


Fig. 3. Password Related Attacks

We roughly describe the attack methods:

- **Brute-force attack:** An adversary can steal the password hash file and recover the passwords from the hash values using brute force computation. He may also use a precomputed dictionary of password hashes [6].
- **Guessing attack:** In practice, most of the users choose weak passwords such that an adversary can find out the passwords of some users of a system by trying common passwords while attempting to login to that system [4], [5]. Spafford suggests a way of choosing good password, in which common words and names are forbidden [13].
- **Network monitoring:** If the communication between a user and a system is unsecured, i.e. unencrypted, an adversary may monitor the network traffic and obtain the passwords or interrupt the traffic while the user creating her password and updating it [12]. This attack is also called man-in-the-middle-attack [1].



1. User sends information, i.e. login request.
2. Adversary may record, modify or block the login request.
User
UNSECURED
Adversary
UNSECURED
Server
4. User sees what adversary sends.
3. Adversary gets critical information.

Fig. 4. Communication over an unsecured channel

- **Phishing attack:** A user can submit her login information to a web page prepared by an adversary, which seems very likely to the original system's login screen. This technique is relatively new, the first attempt was reported in the mid-1990s [15].
- **Malwares:** A Trojan program can capture the key strokes of a victim user and send this information to the adversary [7]. There are some advanced malwares that can steal the login information from messenger like software some of which does not store the login information encrypted [8]. Sun et al. propose *oPass* which uses a user's cellphone and short message service (SMS) to prevent password stealing [14].
- **Visible passwords:** A password that is written to a stickie can be seen by an adversary. He can also watch a user while she enters her password (shoulder surfing). Kumar et al.propose EyePassword, gaze-based password entry, to overcome direct observation [11].

The authors in [9] focus on brute-force attack scenario where an adversary has stolen a file of user names and associated password hashes from the server (see Figure 3). The adversary has also obtained the salt values and other required parameters for computing the hash values. In this scenario, the adversary can make a brute-force attack in order to find one or more users' password (i.e., the adversary can crack most of the hashes.).

The authors in [9] also assume that authentication can only be handled using passwords while logging into the server and the adversary does not compromise the system persistently.

III. Improvements for Honeywords

In this section, we propose three different practical improvements for the honeyword system.

A. Number of honeywords

The authors in [9] recommends a small integer $k = 20$ for the number of honeywords per-user. They note that, though, the number of honeywords does not need to be a system wide parameter. But how do we assess a user's importance and determine an appropriate number for honeywords of her? And how should we maintain this number for each user?

Here we suggest that the number of honeywords per-user should not be a constant parameter. The system should generate more honeywords for users who were attacked before. A user whom honeyword is submitted is more likely to be the target of an adversary than another user whom honeywords are never submitted. The system should reset the password of this user and regenerate her honeywords and update the honeychecker. The honeywords must be renewed after every attack and the number of honeywords must be increased for that user up to a certain amount.

This technique will prevent an adversary to attack the same user again, because this method decreases the success chance of the adversary after each of her unsuccessful attempt.

B. Typo-safe Honeyword Generation

The honeyword generation method called *"chaffing-by-tweaking"* tweaks the selected character positions of the password to obtain a honeyword [9]. This technique is easy to implement on the existing systems since it does not require any change in the login screen. However, since the honeywords differ from the password in a few characters, a legitimate user may submit a honeyword mistakenly and set off an alarm.

There is another honeyword generation algorithm called *"take-a-tail"* which generates honeywords by adding random -generally three digit- integers at the end of the password. As the authors [9] propose, tail tweaking code can be modified so that the difference of two tails is a multiple of a small prime q greater than 10, i.e. $q = 13$.

We generalize this idea to all tweaking methods as follows. After generating a honeyword, a new function $Eval(h, p)$ where h is a newly created honeyword and p is the password of the user, evaluates the typo-safety of the honeyword considering the users keyboard scheme. In this setting, a honeyword which contains a character that is close to, i.e. right or left to, the corresponding character of user's password gets a lower score. If the honeyword's typo-safe score is lower than the minimum allowed score, then the generation procedure generates a new honeyword.

C. Old Passwords Problem

Most users use same password for many services. An old password of a user on some system may be the current password of that user on another system. Thus, taking advanced security measurements may not guarantee the safety. An adversary may attack to a weaker system that the targeted user have an account on it and obtain her old passwords and submit them on a more secure system. This scenario constitutes a security risk.

The authors in [9] give an effective solution where instead of storing old passwords per-user basis the system will store all user's old passwords in a list anonymously. When a password is created, system checks whether this list contains the password. If it is not in the list, the system will not allow that password to be used. However, this solution will not be user-friendly since it is rather strange to forbid to use a password just because of somebody else used it before.

The authors in [9] also propose to encrypt and keep old passwords per-user basis on the actual system and keep the encryption keys in the honeychecker. When needed, the system asks the honeychecker for that user's old passwords key. This seems to be a good solution, however, this method increases the complexity of the system because the honeychecker does more computation, needs more storage and accepts new type of commands which contradicts the simplicity of the honeychecker.

We offer another method to solve this issue. In our solution, instead of encrypting the old passwords the system generates honeywords for old passwords "old-honeywords", as well. The system will generate old-honeywords and keep their hashes with old passwords' hashes per-user basis. There is a probability of that a user may choose a password that is an old-honeyword. But this possibility is negligible.

IV. Discussion and Conclusion

The authors in [9] 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 a 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. The order of the true password is meaningless without obtaining the file of password hashes.

On the other side, honeyword system is not a complete solution for the password management problem. The following scenarios should also be considered:

- An adversary can infect the whole system, and learn the index of real password among sweetwords of a user.
- An adversary can steal the sweetwords of a user and submit on another systems which does not use honeywords.

In this work, we revisited the paper [9] and suggest some possible improvements for

- number of honeywords of a user.
- generating typo-safe honeywords.
- managing old passwords.

We hope our contributions improve honeywords technique and help designing more secure systems.

References

[1] National information assurance (ia) glossary, 2010.

[2] Password cracking. Web Site, 2013. www.golubev.com/hashgpu.htm.

[3] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *ESORICS*, pages 286–302, 2010.

[4] J. Bonneau. Guessing human-chosen secrets. Technical Report UCAM-CL-TR-819, University of Cambridge, Computer Laboratory, May 2012.

[5] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *IEEE Symp. Security and Privacy*, 2012.

[6] A. Conklin, G. Dietrich, and D. Walz. Password-based authentication: A system perspective. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 7 - Volume 7*, HICSS '04, pages 70170.2–, Washington, DC, USA, 2004. IEEE Computer Society.

[7] D. Elser and M. Pekrul. Inside the password-stealing business: the who and how of identity theft, 2009.

[8] J. Erasmus. Malware attacks: Anatomy of a malware attack. *Netw. Secur.*, 2009(1):4–7, Jan. 2009.

[9] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. Unpublished draft.

[10] 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*, pages 523–537, 2012.

[11] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *Proceedings of the 3rd symposium on Usable privacy and security*, SOUPS '07, pages 13–19, New York, NY, USA, 2007. ACM.

[12] P. G. Neumann. Risks of passwords. *Commun. ACM*, 37(4):126–, Apr. 1994.

[13] E. H. Spafford. Opus: preventing weak password choices. *Comput. Secur.*, 11(3):273–278, May 1992.

[14] H.-M. Sun, Y.-H. Chen, and Y.-H. Lin. opass: A user authentication protocol resistant to password stealing and password reuse attacks. *Information Forensics and Security, IEEE Transactions on*, 7(2):651–663, 2012.

[15] A. van der Merwe, M. Loock, and M. Dabrowski. Characteristics and responsibilities involved in a phishing attack. In *Proceedings of the 4th international symposium on Information and communication technologies*, WISICT '05, pages 249–254. Trinity College Dublin, 2005.

[16] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 391–405, Washington, DC, USA, 2009. IEEE Computer Society.