Examination of a New Defense Mechanism: Honeywords

Ziya Alper 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, provide some possible improvements which are easy to implement and introduce an enhanced model as a solution to an open problem.

***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 find 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 I, the system adds honeywords hashes to this file at random orders. Thus an adversary who has cracked the password hashes 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	$\mathcal{H}(p_4)$

→

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 login server will determine the order of her among the users, and the order of the submitted password among her sweetwords. The login 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 Figure I. The honeychecker cannot know anything about the user's password or honeywords. It maintains a single database that contains only the order of the true password among the user's sweetwords.

The adversary can steal the file of hashed passwords and invert the hashes but cannot tell which sweetword

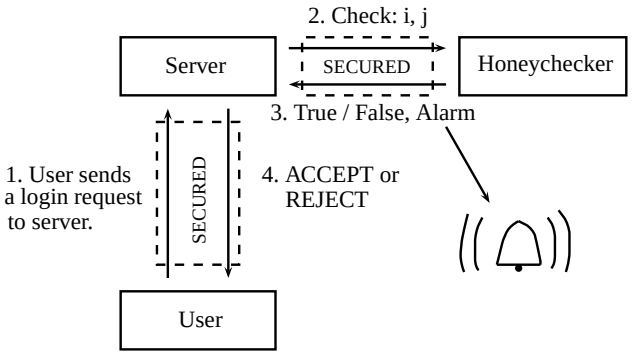

Fig. 2. Login schema of a system using honeywords.

is the password. There will be a risk of detection that prevents the adversary to login to the system.

A closely related work is the Kamouflage system of Bojinov et al. [3] though that work differs from honeywords. In that system, the user's password list is placed with another lists that contains honeywords. There is no need for a server in Kamouflage system although the authors in [3] note that servers might be used to empower the ability of detection of compromise.

Our contribution. In this work, we analyze the honeyword system according to both functionality and the security perspective. Then, we suggest improvements for number of honeywords per user, generating typo-safe honeywords and managing old passwords. Finally we introduce an enhanced model of honeywords which may be a solution to the open problem of active attacks.

The rest of the paper is structured as is follows: Section II briey explains the password related attack models. Section III describes the improvements that we suggest for the proposed honeywords system in [9]. In Section IV, we introduce an enhanced honeywords system. Section V concludes the paper.

II. Attack Models

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

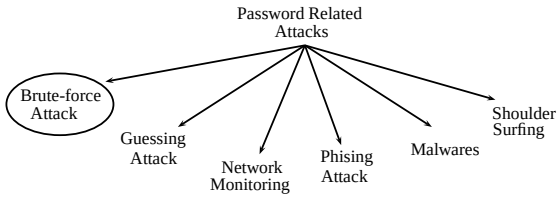

Fig. 3. Password Related Attacks

Password attacks can be classified as follows:

- **Brute-force attack:** An adversary can steal the password hash file and crack the hashes using brute force computation.He may also use a precomputed dictionary of password hashes [6].
- **Guessing attack:** Many 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 suggest good password choice should avoid common words and names [13].
- **Network monitoring:** If the communication between the user and the system is unsecured, i.e. unencrypted, an adversary may monitor the network traffic and obtain the passwords or interrupt the traffic while a user creating her password and change it to another one [12]. This attack is also called man-in-the-middle-attack [1].

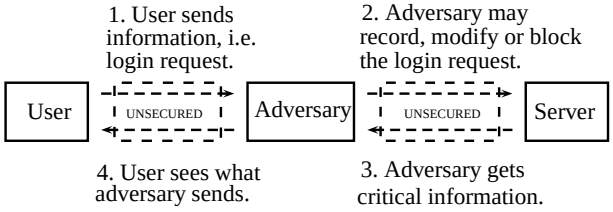

Fig. 4. Communication over an unsecured channel

- **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 and send this information to the adversary [7]. There are some advanced malwares that can steal the login information from messenger like softwares some of which does not keep 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 II). The adversary has also obtained the salt values

and other required parameters for computing the hash function. In this scenario, the adversary can make a brute-force search to find one or more user's 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

We are now ready to propose our practical improvements for the honeyword system. n this section, we present four distinct solutions. The first three solutions are proposed to make the system more robust. Our last solution is related to a problem mentioned in [9], i.e., we deal with an active attack scenario to 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?

Instead of having constant number of honeywords per-user, the number of users' honeywords should be dynamic once there is an active attack. Namely, the system should generate more honeywords for users who were previously attacked. Our suggestion comes from the following fact: A user whom honeyword is submitted is more likely to be the target of an adversary than users whose honeywords are never submitted. The system should be setup in such a way that the password of this user is reset and her new honeywords are regenerated and honeychecker is updated accordingly. The honeywords should be renewed after every attack, and in order to decrease the probability of an attacker the number of honeywords of that user should be increased up to a certain security level.

This technique will deter the adversary to attack the same user again, because the success chance of the adversary will be much lower.

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 then the minimum allowed score, then the generation procedure generates a new honeyword.

C. Old Passwords Problem

Most users use same password on different systems. 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 a old-honeyword. But this possibility is negligible.

IV. A Solution to an Open Problem: Active Attacks Against Honeywords System

The honeywords system is only designed to withstand off-line attacks. In this scenario, we assume, as the authors mentioned in [9], that the adversary has only stolen the password hashes but did not compromise the system on a persistent basis, i.e, the adversary did not hack the system or did not gain the admin rights. However, the authors in [9] mentioned about a problem which we believe is still open: How can a honeyword system be best designed to withstand active attacks, e.g., code modification, of the system (or the honeychecker)?

The question is very reasonable as the adversary who accesses the password hashes may also gain other permissions like administrator rights. In this case, the system is corrupted and therefore may behave arbitrarily.

A. Assumptions

In our proposal, we assume that the login server and the honeychecker cannot be compromised at the same time (otherwise, the honeyword scheme will not be secure at all). We also assume that the administrators of login server and honeychecker do not cooperate.

B. Adversarial Capabilities

In our model, we classify adversarial attack scenarios into two classes.

- The adversary has compromised the login server and has gained administrator rights. He can now modify the codes in the login server as well as other system components.
- The adversary has compromised the honeychecker. He can now modify the codes of the honeychecker as well as other system components.

In the first attack, the attacker has gained administrator rights and has system wide effects. Thus she can send any message (or request) to honeychecker. Note that the honeychecker understands only two type of messages: **Set** or **Check** which sets or checks the order of a user's password among her honeywords, respectively. The adversary can send a **Set** message to the honeychecker and set the order of a user's password to a value whatever he wants.

In the second attack, the adversary has administrator rights on the honeychecker. In this case, the most important attack that the adversary can do is DoS attack. The attacker can disallow any legitimate login request or can allow any illegitimate login attempt. The attacker may also modify the honeychecker to send True of False results at random.

C. The Proposal

The honeychecker described in [9] trusts the login server and does not validate any **Set** message. However, in one of our attack scenarios the login server is malicious. Thus we need to enhance the Honeychecker in such a way that it can validate the origin of **Set** messages. The login server sends **Set** messages to honeychecker if:

- A user signs up, i.e. she creates her password for first time.
- A user changes her password.

Since the login server is compromised, the validation process cannot be done on it. We need a secure channel to communicate with the user whose password's order is being set but honeychecker does not know any information of the user. One of the design principles of honeychecker is that compromise of only the honeychecker does not reduce the security level of the whole system lower than before introducing the honeywords. Hence we should give honeychecker minimal information about the user and this information must be enough to communicate with the user to validate her with a fair confidence. In todays world, the most common way of this communication is done on Short Message Service (SMS).

In our model, if a user signs up, the login server asks her to enter personal information including mobile number and updates the honeychecker. Here we add an extra parameter to **Set** message: *phn*, the users mobile number. The honeychecker will add this information to its database and will communicate with the user when needed. The sign up scheme of an enhanced honeyword system is depicted in Figure IV-C.

If a user would like to change her password, she will send a password change request to login server. The login server will send a **Set** message to honeychecker. The honeychecker now knows the mobile number of the user and asks her to validate the origin of the request. The honeychecker will set an alarm if a lot of invalid update

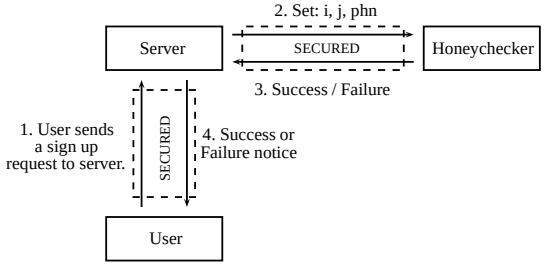

Fig. 5. Sign up schema of an enhanced honeywords system.

requests are made. The alarm may be an e-mail but this mail must be sent to a server other than the login server, since it is compromised. The password change scheme of an enhanced honeyword system is depicted in Figure IV-C.

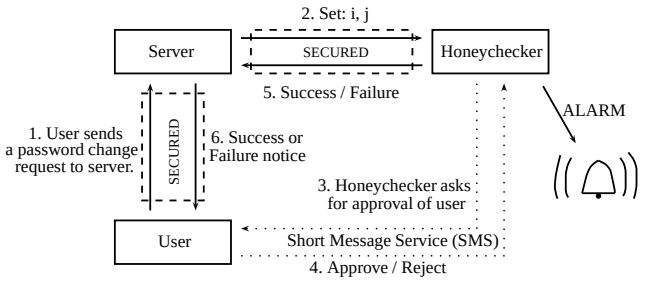

Fig. 6. Password change schema of an enhanced honeywords system.

D. Security Analysis of New Model

If the adversary gains administrator rights on login server, she can send **Set** messages which she desires. In this case, the honeychecker will confirm the update request by asking the user. A dishonest login server will fail to change a user's order of password in this scenario.

The adversary can also send repetitive **Check** messages to find the order of a user's password. We suggest that the honeychecker counts and monitors the check requests and decides whether the login server is compromised or not. High number of check requests means that the (malicious) login server is making a brute-force search to find the correct order of password of all users.

An adversary may also attack honeychecker and may gain administrator rights on it. She can modify the honeychecker to send arbitrary results to login server after **Check** messages. Our suggestion for fighting with this attack is creating some small number of dummy accounts to test the honesty of the honeychecker. The login server will send valid login information with half of the accounts and invalid login information with the other half frequently. If the honeychecker is infected, i.e. compromised, it will not take the correct action. Thus the adversary will be detected.

Our enhanced model of honeyword system is more robust to active attacks than the primitive system designed in [9].

V. 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 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 examined the paper [9] and suggest some possible improvements for

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

and introduced an enhanced honeywords system which may be a solution to the active attacks problem.

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.