Paper

Detecting Password File Theft using Predefined Time-Delays between Certain Password Characters

Khaled W. Mahmoud, Khalid Mansour, and Alaa Makableh

Department of Computer Science, Zarqa University, Zarqa, Jordan

https://doi.org/10.26636/jtit.2017.112517

Abstract—This paper presents novel mechanisms that effectively detect password file thefts and at the same time prevent uncovering passwords. The proposed mechanism uses delay between consecutive keystrokes of the password characters. In presented case, a user should not only enter his password correctly during the sign-up process, but also needs to introduce relatively large time gaps between certain password characters. The proposed novel approaches disguise stored passwords by adding a suffix value that helps in detecting password file theft at the first sign-in attempt by an adversary who steals and cracks the hashed password file. Any attempt to login using a real password without adding the time delays in the correct positions may considered as an impersonation attack, i.e. the password file has been stolen and cracked.

Keywords—access control, intrusion detection systems, network security, password protection.

1. Introduction

The wide spread of communication methods, especially through the Internet expedites various computational tasks and facilitates data sharing amongst people and organizations. Several methods were proposed in the literature to enhance data security and protect privacy. Customers' passwords are amongst the most important data to secure. According to what was recorded in the last few years about password file thefts, obtaining the maximum level of protection to secure password files and detecting password file thefts is still a major challenge in the digital system world [1].

As textual passwords are still widely used as an authentication mechanism and they are unlikely to disappear in the near future [2], [3], passwords should be selected and saved carefully. Many password file theft incidences were reported recently [4], [5], and big companies have been exposed to password database theft such as Yahoo, Hotmail, Social Security Administration [1]. This urges security community to develop methods to increase the immunity of passwords and effectively detects the theft of password files. Moreover, timely discovering the theft of password files is a very important issue. Storing passwords properly is a key issue in protecting them.

This paper proposes adding an extra level of security that puts the adversary at risk of being detected at the first login attempt. This helps companies to deny any access from adversary who is trying to impersonate users and access the system as a legitimate user. The proposed mechanisms are built on the password hardening technique that was presented in [6] and [7]. This password hardening technique depends mainly on the way users perform the sign-up step. When a user leaves relatively large time gaps between certain consecutive password characters, an adopted keying pattern is created and associated with the password. It allows to both harden the password and detect password file thefts.

The rest of this paper is organized as follows. Section 2 presents the common password file structures. Section 3 reviews the related work. In Section 4, the idea of password pattern is explained. Section 5 shows the proposed approaches for detecting password file theft. Finally, research conclusions and suggestions for further studies are discussed in Section 6.

2. Password File Structure

Password-based authentication systems require password files or tables to organize users' login information. In this critical file, which should be kept secret, users' IDs are stored in plaintext. However, the passwords are stored according to the security policy adopted by the used system. Three types of security policies exist:

- Plain text – some servers store passwords as a plaintext. In this case, security depends entirely on the password file secrecy (i.e. protecting the file from being stolen). Once the authentication server is compromised and the password file is stolen, the adversary can impersonate users and logs in using the correct passwords.
- Hashed passwords – hash function (or algorithm) is a function that can be used to transform the input data into a fixed size output (hash value) regardless the input size. This function is a one-way function because it is easy to get the result for any given input but it is very difficult or impossible to get the input backward given the result and the function. Many hash functions exist and some have many versions

such as: SHA1, SHA256, SHA512, MD2, MD4 and MD5. In this type of security policy, the result of hashing the original password $H(P)$ is stored in the password file. Unfortunately, this way is not sufficient against brute force and dictionary attacks. In case the list of hashed passwords is stolen, the attacker can apply offline brute-force attack to find a password with a hash value equivalent to the value stored in the stolen list. Later on, the adversary can impersonate the user and logs in using the correct password.

- Hash + salt – salt is a random data that is appended to the password before passing it to the hash function $H(p||s)$. Salt is used to increase the amount of cracking time by increasing the size of passwords space. In addition, it is used to overcome the fact that many user's accounts have the same password and so their hash values. Despite this enhancement, passwords are still under leakage by attackers.

3. Literature Review

Several techniques are proposed in the literature to either harden the cracking process or to discover the theft as soon as possible.

Injection "deceive" in password-based systems is a general trend that aims to hide the correct passwords and to enable fast discovery of password file thefts. For example, some approaches suggest to insert fake accounts in the passwords file or insert extra fake password files [8], [9], any use to those accounts or files indicates an intrusion [10]. More about those approaches are given next.

3.1. Kamouflage

Kamouflage is proposed as an architecture for building theft-resistant password managers [11]. It aims to protect local password database (e.g., stored on a laptop or a cell phone) by adding a list of statistically indistinguishable decoy passwords to hide the real list. Decoy sets are generated using certain rules based on the correct passwords. In case the attacker decrypts passwords, it is difficult for attackers to distinguish between the correct password set and the decoy sets.

3.2. Honeywords

Honeywords are sets of decoy passwords that are added to each user accounts to deceive the adversary [10]. Honeywords, which are similar to user-selected passwords, are stored in hashed form along with the correct password, i.e. the structure of the password file is altered by adding a list of candidate passwords for each user rather than a single password for each account. In this case, the adversary faces a list of candidate passwords for each user and this makes the process of distinguishing between honeywords and the correct user's password harder for the adversary who steals and compromises a password file. Consequently, using any of these honeywords to login will generate an alarm to the system administrator. This alarm implies that the passwords file was stolen and cracked and an illegal login attempt is recorded.

The honeywords system requires an extra secure server called honeychecker to store the indexes of correct passwords. The server is connected to the main server through a dedicated line. This server aims to improve the overall security by separating the information related to users' authentication process. During the login process, if the entered password is not in the list associated with the user then the login failed. If a match is found, then the index of this match is sent to the honeychecker for verification. If the index is correct, then the user is authenticated. Otherwise, the honeychecker will raise a silent alarm to the system administrator and return false to the main server to deny the login.

To ensure the effectiveness of the honeywords approach, honeywords should be generated in a manner that makes every sweetword (sweetwords list is a set of all honeywords plus the correct password) to seem strongly as a candidate password. This makes it difficult for the adversary to guess the correct password. In addition, if a wrong password is used, a password alarm can be raised by the system. Different methods to generate honeywords can be found in [10]. Some of these methods can be done without users' intervention while others require user intervention (i.e. by modifying the sign-up interface). More methods can be found in [12], [13].

An enhancement that was presented in [14] aims to increase the effectiveness of the honeywords system and to overcome more active attack scenarios by adding phone number as an extra information field for each user in the honeychecker database. Honeychecker uses phone number to communicate with the user in special cases such as password change or many invalid sign-in operations.

3.3. Paired Distance Protocol

The article in [13] proposed a new honeywords system based on paired distance protocol (PDP) with less storage overhead. In PDP, the user chooses a random string (RS) that will be appended to the password during registration. A distance chain – which is derived from RS – is used to replace the honeywords for each user account. Distance chain is a set of paired distances (separated by "–") between every two consecutive elements of RS. This distance is calculated using a secret honey circular list (hcl) that holds the keyboard letters and digits spread in random order (see Fig. 1). As an example, the distance chain for the random string $fa5$ is (20–22), i.e. the number of elements that is traversed in clockwise between f and a in hcl is 20 and the number traversed in clockwise between a and 5 is 22.

The password file contains the username, the hashed password and the distance chain for the RS. The honeychecker is still needed to store the username and the first character of the RS. During login, if the user submits an incorrect

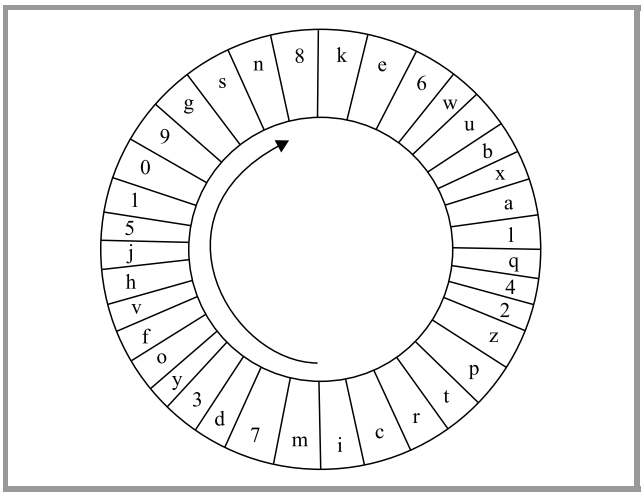

Fig. 1. Random order for 36 letters and digits in honey circular list.

password, then the system will reject the user login directly. If the password is correct, then the system will calculate the distance chain from the given RS and compare it with the stored distance chain in the password file. If no matches are found, then the login will be denied. If the derived distance chain gets matched with a stored one, then the system will match the first character in the given RS with the character stored in the honeychecker. If a match is found then the login is permitted. The success of PDP is associated with the high randomness in choosing RS. PDP provides security against multi-system attacks and DoS.

3.4. Two Password Files Model

The paper in [12] proposed a new system that simulates the honeywords system. However, the difference is that the proposed mechanism stores the user's authentication information in two password files F1 and F2 that are located in the main server.

- F1 is a two-column table. The first column is a list of all usernames sorted in an alphabetical order. A list of indexes that are randomly selected from the first column in F2 is stored in the second column beside each user. One of these indexes is the index of the correct password. The correct index is paired with the username and delivered to honeychecker.
- F2 is a two-column table. The first column is just an index column and the second column is the hash value of each account's password. Some of these accounts are decoy user accounts (or honeypots). They are created by the administrator during system initialization.

During user login, the system authenticates the user as follows:

1. The system goes through F2 in order to find the index of the given password.
2. If this index is not in the corresponding list of this user in F1 then the login fails directly.
3. Otherwise, the system checks if the returned account is a honeypot account. If yes, the system will raise an intrusion alarm.
4. If this account is not a honeypot account, then the returned index is sent to the honeychecker to check if it is the correct index. If no, the honeychecker will again raises an intrusion alarm.

3.5. ErsatzPasswords

ErsatzPasswords scheme is another solution provided in [4] to control the problem of password file theft and detect the leakage attempts. The proposed scheme uses a machine-dependent function (HDF) at the authentication server to harden the off-site password discovery. Adding this hardware supports the software system and makes the cracking process impossible without accessing the target machine physically. Moreover, a single password file identical to the traditional password file exists and no additional server is needed. Whenever the attacker attempts to crack the password file offline, he/she uncovers ersatzpasswords, i.e. fake passwords, which will raise an alarm in case of their use. The following steps are used during sign-up operation:

1. Each user presents his username, password p and ersatzpassword p^*.
2. Calculate the salt $s^* = \mathbf{HDF}(p) \oplus p^*$.
3. Calculate $B = H(\mathbf{HDF}(p) \oplus s^* || s^*)$.
4. B and the associated username are stored only in the password file.

During the login process, if the presented password is correct and B is equal to the stored value then the user is successfully authenticated. If the presented password was the ersatzpassword p^* (i.e. the password file was stolen and the hash values were inverted), then the result of $H(p^*||s^*)$ will be equal to the stored B and consequently an alarm will be raised. Finally, if there is no match, then this can be treated as an error login.

3.6. PolyPasswordHasher

PolyPasswordHasher is a software only solution proposed in [5] that reduces the possibility of cracking user's passwords by making them interrelate. The basic idea in this mechanism is to XOR a secret share with a salted password hash at the account creation time, where Shamir secret sharing [15] method is used to create the shares. The result of XORing is stored in a password file given that neither the share nor the salted password hash exists on the disk. To specify which share was used, an extra field called the share number is added to the password file. This scheme prevents the adversary from validating the hash value for password,

and the attacker cannot gain any information from compromising the password file because the share and the password hash value are not stored on the disk. However, if the adversary reached the arbitrary memory on the server then he can steal the secret.

3.7. Synergetic Authentication

The study in [16] proposed Synergetic Authentication (SAuth) protocol. A user can be authenticated by a site if and only if true credentials are used at that site and another vouching site voted for authenticating that user. For example, let user A has an account on the system S and also has an account on the system V. The login request of A to access S will not be accepted unless the system V sends vouching message to system S ensure the truth of A information. So, if an adversary compromised the database of the system S and tries to impersonate the user A his try will fail. The adversary must compromise both systems S and V simultaneously to success in this attack.
The design of SAuth system provides a mechanism that can detect the activities that seem irregular. If the vouching system received several user authentication requests ended with fail, this could be considered as an attempt to impersonate a certain user(s). The major limitation to this mechanism is when users have the habit of password reusing. To alleviate this risk, decoy passwords are proposed. The other limitation is the need for synchronizing between different services.

4. Hardening Passwords by Adding Delays between Keystrokes

Text passwords are common, easy and reliable authentication method. To ensure the desired goal of access protection, the creation of passwords and how they are stored need to be managed carefully. Several research papers were dedicated to harden passwords and make them stronger. Some of these methods verify not just the knowledge of the password, but also verify other credentials such as: the possession of a specific token, the current GPS location of the user [17], the fingerprints [18], the signature or something individuals can be characterized with such as keystroke dynamics biometrics [19], [20].
In this paper, a modified version of the method that was proposed in [6] to strengthen passwords is adapted and used in detecting passwords file theft. In Mansour's method, users need to type their passwords in certain rhythms during the sign-up process. During sign-up, the user is required to enter his password twice. The password is accepted only if the two trials return the same keying pattern. Otherwise, the user is instructed to repeat entering the password twice again. For a successful login, a user needs to key password with the same sign-up rhythm. A strong password is formulated based on classifying the delay times between consecutive password characters into either slow or fast based on a certain threshold value. The exact delay time values are not critical since the keying rhythm is classified into fast and slow regardless the exact delay times.
In the modified version and in order to simplify the registration process, the user is deliberately declaring the location(s) of large delay. The users are instructed to enter their passwords normally without leaving large delays and then select – for example by mouse – the large-delay location(s). A special sign-up interface is designed and some components (such as check boxes) to specify the delay positions are added. At sign-in, the login is accepted only if the user enters the correct password with large delay in the specified position(s). The user should leave a delay time at the selected position(s) that exceeds normal typing. As an illustration, if a user specifies a delay after the third character, then the system will not allow the user accessing the system unless he/she types in the correct password and leave relatively large time delay after the third character. Any accidental waiting time during typing password will be considered as a large delay and will lead to reject the login if this delay is in the wrong position. For more details refer to [7].
It should be clear that this approach of password hardening is different from password hardening based on keystroke dynamics. Authentication systems based on keystroke dynamic capture normal typing behaviors of users using long training session(s). In these systems, the users are authenticated only if they write the correct password using their way of writing (i.e. rhythm) [21]. In proposed system, users predefined the keystroke pattern rather than using his human patterns. Consequently, proposed system does not require training or analyzing the keying behaviors of users. In addition to hardening textual passwords, the above idea is extended to detect a possible password file theft. The next section shows how the adopted keying pattern data can be used to detect password file theft.

5. Password File Theft Detection

According to what was recorded in the last few years about password file thefts, obtaining the maximum level of protection to secure password files and detecting any violation over the password file is still a challenge in the digital system world [1]. In this section, three novel password file theft detection mechanisms are presented. The first two mechanisms require an auxiliary secure server, while the third one uses login server only.

5.1. Preliminaries

The following notations are defined for the i-th user: u_i is the username, p_{ij} is a honeyword number j that belongs to user i. The honeywords plus the correct password p'_i forms what we call sweetwords. k is the number of sweetwords, c_i is the index of the correct password for user i, and dp_i is the delay pattern. The delay pattern is a sorted comma-separated list of all positions where user i should leave

large delay. For example, if the large delays come after the fourth, the second and the seventh character, then the delay pattern is $<2,4,7>$. Finally, the *delaylist* is a list of time delays between every two successive password characters.
In proposed system, two types of servers can be used: login server and auxiliary server (or honeychecker). The login server maintains the password file while the auxiliary server maintains the authentication file, which stores data related to the correct passwords. We assume that a dedicated line for communication between honeychecker and the login server is used. In addition, both types of servers are able to raise an alarm if a password file disclosure is detected.

5.2. Honeychecker Server-based Mechanisms

In the first mechanism a traditional password file structure is used to store usernames and the hash value of correct passwords. The honeychecker server stores users' names and their associated delay patterns *dp*.
In case the password file was hacked and cracked, the system makes it more difficult for the adversary to access the system since the password has to be keyed in a certain pattern given that this pattern does not exist in the login server. With every accessing attempt, if the entered password was correct, its keying pattern *user-dp* is extracted and sent to the honeychecker server. If the pattern is correct, then it is considered a successful attempt, otherwise it is considered a possible password file theft, see Algorithm 1.

Algorithm 1. Login and user authentication algorithm for the first mechanism

```
u ← read the entered username
(p, delaylist) ← read the entered password and record
the delays between each successive character.
search the password file for the user u
if u is not stored in the password file then
   return login-fail
else
   p' ← get the correct password from password file
end if
if p ≠ p' then
   return login-fail
else
   user-dp ← extract-the-delays (delaylist)
   send u and user-dp to the honeychecker
   stored-dp ← get the stored delay pattern for this user
   from honeychecker
   if user-dp = stored-dp then
      return login-succeed
   else
      raises a theft alarm
   end if
end if
```

To extract the delay pattern from the collected delaylist, a detailed algorithm is given in [7]. It sorts the list of delays in an ascending order a long with the original position of each delay value. Then it computes the difference between every two-successive delay in the sorted list and returns the location of the maximum difference. All elements appear after this location is considered the user delay pattern.
This mechanism is characterized by its simplicity and effectiveness since neither extra storage space is needed to store honeywords nor extra effort is needed to design these words.
If the password was only correct, we cannot be sure that this case is a theft of password file since a legitimate user can enter his password without its correct pattern for various reasons such as fatigue and forget cases (false positive). However, when the system raises an alarm for possible file theft, the administrator needs to take further actions to check the current situation.
To further protect passwords in case of password file theft, the next mechanism augments honeywords with correct password in the password file.
The second mechanism uses password file with honeywords. With this mechanism, each user account in the password file is associated with a list of honeywords as shown in Table 1. Note that, the sweetwords are stored in the password file without any information related to the keystroke latencies.

Table 1
Passwords file structure stored in login server based on honeywords system

Username	Sweetwords
u_1	$\{p_{11}, p_{12}, p_{13}, p'_{14}, \ldots, p_{1k}\}$
u_2	$\{p_{21}, p_{22}, p_{23}, p_{24}, \ldots, p'_{2k}\}$
$\ldots$	$\ldots$
u_n	$\{p_{n1}, p'_{n2}, p_{n3}, p_{n4}, \ldots, p_{nk}\}$

The honeychecker (auxiliary secure server) is required in the authentication process. In addition to the username and the delay pattern *dp* a new column is added in the authentication file to store the index c_i of the correct password p'_i in the list of sweetwords (see Table 2).

Table 2
The authentication file structure stored in honeychecker

Username	Index (c_i)	Delay pattern (dp_i)
u_1	c_1	dp_1
$\vdots$	$\vdots$	$\vdots$
alice	3	3,6
$\vdots$	$\vdots$	$\vdots$
u_n	c_n	dp_n

As any normal login process, the user enters his credential information, i.e. username and password. The system captures username u, password p and the keystroke laten-

cies between every two successive characters. Firstly, the system checks whether the entered password p is in the sweetwords list. If p does not exist then the login is denied. Otherwise, the index of this sweetword (call it y) is sent to the honeychecker paired with the username u and the users' delay pattern that was extracted from delaylist. The honeychecker checks whether y matches the stored index c for this user. If no match is found then the honeychecker raises a theft alarm, else the honeychecker checks for the correctness of the delay pattern. If the password p is entered correctly (i.e. with relatively large time gap in the specific positions), the honeychecker returns a message to accept the login, otherwise a theft alarm is raised. This process is shown in Algorithm 2.

Algorithm 2. Login and user authentication algorithm for second mechanism

```
u ← read the entered username
(p, delaylist) ← read the entered password and record
the delays between each successive character.
search the password file for the user u
if u is not stored in the password file then
  return login-fail
else
  sweetwords ← get the list of sweetwords from login
  server
end if
if p ∉ sweetwords then
  return login-fail
else
  y ← get the index of p in sweetwords
  user-dp ← extract-the-delays (delaylist)
  send u, y and user-dp to the honeychecker
  (c, stored-dp) ← search the authentication file for the
  user u and return the index of the correct password
  and its delay pattern.
  if y ≠ c then
    raises a theft alarm
  else
    if user-dp = stored-dp then
      return login-succeed
    else
      raises a theft alarm
    end if
  end if
end if
```

According to the honeywords system presented in [10], when the attacker gets the password file and inverts the hashed values, any use of honeywords (wrong password) will raise a theft alarm. However if accidentally the correct password is picked, the attacker gets in. In proposed mechanism, a new layer of protection has been added. If a user accidentally tries the correct password without keying it according to the predefined delay pattern, the system raises a theft alarm. Consequently, the chance for password file theft detection is increased.

To have a more secure authentication system, the number of sweetwords per user suggested in [10] was 20 and can reach 1000 in some circumstances. In proposed mechanism, the number of sweetwords can be much less due to the fact that guessing the correct password does not guarantee successful access to the system. If the attacker was able to guess the correct password due to the existing small number of sweetwords, the second piece of information (i.e. delay pattern) gives a second layer of protection. Moreover, the problem of designing honeywords become easier for small set of sweetwords.

5.3. Login Server-based Mechanism

Some authentication systems suffered from increasing the required storage area such as adding $k-1$ honeywords for each user as in [10], or adding additional file as in [12]. The storage cost is increased as the number of users in the system increases. Hence, storage optimization becomes an issue. Moreover, an auxiliary server is also needed in the authentication process. Using this server is considered an extra storage cost. Eliminating this server may simplify the authentication process. In this section a new mechanism that requires no honeychecker server and no honeywords is introduced, where each password by itself works as a password and a honeyword at the same time. The trap is in a single password rather than a list of honeywords.

The passwords file (see Table 3) has the same structure as in traditional passwords file. It contains two columns of information: usernames and hashed passwords $h(uas_i)$ where *uas* stands for "user authentication string". The user authentication string is actually a concatenation of two parts: p'_i and dp_i. For example, if user i selects his password as me@me12 and the delay pattern was $<2,5,7>$, then uas_i= me@me12257.

Table 3
Password file structure

Username	Password
u_1	$h(uas_1)$
u_2	$h(uas_2)$
$\vdots$	$\vdots$
u_n	$h(uas_n)$

During the login process, the user enters his credential information (username and password). The password should be keyed in the predefined pattern. The first authentication step checks if the entered password p is equal to *uas*. If yes, raise a password file theft alarm. This step adds a security level that aims to detect password file theft and cracking as early as possible. If there are no penetration signs, the system extracts the user delay pattern from the delaylist and checks whether *h(p || user-dp) = h(uas)*. if yes then the login is succeeded otherwise the login is failed. This process is shown in Algorithm 3.

This mechanism is a simple one with no extra storage overhead; no need to k honeywords. Clearly, if the system

has n users then no more than n authentication values are stored. Moreover, no auxiliary server is needed. Additionally, it has a normal password file structure. Such normality makes the adversary believe that the cracking results are original passwords and this increases the possibility of trapping him.

Algorithm 3. Login and user authentication algorithm for the third mechanism

```
u ← read the entered username
(p, delaylist) ← read the entered password and record
the delays between each successive characters.
search the password file for the user (u)
if u is not stored in the password file then
   return login-fail
else
   h(uas) ← get the stored hashed password
end if
if h(p) = h(uas) then
   raises a theft alarm
else
   user-dp ← extract-the-delays (delaylist)
   if h(p || user-dp) = h(uas) then
      return login-succeed
   else
      return login-fail
   end if
end if
```

Finally, to further disguise the cracked passwords, dp can be encoded into alphabets and inserted in a position that depends on the average of dp. The insertion positions can vary for different users depending on their delay patterns.

6. Conclusion

The proposed mechanisms use a new strong password that is based on augmenting time delays between certain password characters. These delays are used in both accessing systems and detecting password file thefts. Two of the proposed methods use a honeychecker server while the third one applies a different mechanism that does not need a honeychecker server.

The three mechanisms can raise a theft alarm. Particularly, in case of correct password and wrong delay pattern. This can happen in two cases: the first one is when the attacker succeeds in getting the password and tries guessing the delay pattern, while the second case is when legitimate users do certain mistakes in the pattern during the login process. Many actions can be taken in these cases such as blocking the account or sending an alarm to the genuine user. In order to avoid raising an alarm for legitimate users, the system can be tuned such that the system can raise a silent alarm to the administrator for the first failed login but after a few failed login attempts, the system can take stronger actions.

As a future work, large scale empirical study is needed to analyze both the effectiveness of the proposed password hardening system and the ability of the three proposed mechanisms in detecting password file theft.

Acknowledgements

This research is funded by the deanship of scientific research at Zarqa University, Jordan.

References

[1] D. Mirante and J. Cappos, "Understanding Password Database Compromises Technical Report", Tech. Rep. TR CSE-2013-02, Polytechnic Institute of NYU, 2013.

[2] D. Florêncio, C. Herley, and P. C. van Oorschot, "An administrator's guide to internet password research", in *Proc. 28th Large Instal. Sys. Administr. Conf. LISA14*, Seattle, WA, USA, 2014, pp. 44–61.

[3] P. Jadhao and L. Dole, "Survey on Authentication Password Techniques", *Int. J. of Soft Comput. and Engin. (IJSCE)* vol. 3, no. 2, pp. 67–68, 2013.

[4] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford, "Ersatzpasswords: Ending password cracking and detecting password leakage", in *Proc. of the 31st Ann. Comp. Secur. Appl. Conf. ACSAC 2015*, Los Angeles, CA, USA, 2015, pp. 311–320.

[5] J. Cappos and S. Torres, "PolyPasswordHasher: Protecting Passwords in the Event of a Password File Disclosure", Tech. Rep., 2014 [Online]. Available: https://password-hashing.net/submissions/specs/PolyPassHash-v1.pdf

[6] K. Mansour, "Adopted keystroke rhythm for password hardening", in *Proc. 11th Int. Conf. on Passwords Passwords 2016*, Bochum, Germany, 2016.

[7] K. W. Mahmoud, "Elastic password: A new mechanism for strengthening passwords using time delays between keystrokes", in *Proc. 8th Int. Conf. on Inform. and Commun. Syst. ICICS 2017*, Irbid, Jordan, 2017, pp. 316–321.

[8] M. J. A. Mohammed Almeshekah and Eugene H. Spafford, "Improving Security using Deception", Tech. Rep. 203-13, Center for Education and Research Information Assurance and Security, Purdue University, West Lafayette, USA, 2013.

[9] F. Cohen, "The use of deception techniques: Honeypots and decoys", *The Handbook of Inform. Secur.*, vol. 3, no. 1, pp. 646–655, 2006.

[10] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable", in *Proc. of the 20th ACM SIGSAC Conf. on Comp. and Commun. Secur. CCS 2013*, Berlin, Germany, 2013, pp. 145–160.

[11] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, "Kamouflage: Loss-resistant password management", in *Proc. 15th Eur. Symp. on Res. in Comp. Secur. ESORICS 2010*, Athens, Greece, 2010, vol. 6345, pp. 286–302.

[12] I. Erguler, "Some remarks on honeyword based password-cracking detection", *IACR Cryptology ePrint Archive*, vol. 2014, p. 323, 2014.

[13] N. Chakraborty and S. Mondal, "A new storage optimized honeyword generation approach for enhancing security and usability", *Comput. Res. Repository*, vol. abs/1509.0, p. 8, 2015 (arXiv:1509.06094).

[14] Z. A. Genc, S. Kardas, and M. S. Kiraz, "Examination of a New Defense Mechanism: Honeywords", *IACR Cryptol. ePrint Archive*, vol. 2013, p. 696, 2013.

[15] A. Shamir and A. Shamir, "How to share a secret", *Commun. of the ACM (CACM)*, vol. 22, no. 1, pp. 612–613, 1979.

[16] G. Kontaxis, E. Athanasopoulos, G. Portokalidis, and A. D. Keromytis, "Sauth: Protecting user accounts from password database leaks", in *Proc. of the 20th ACM SIGSAC Conf. on Comp. and Commun. Secur. CCS 2013*, Berlin, Germany, 2013, pp. 187–198.

[17] N. Abdelmajid and K. W. Mahmoud, "Global position system location-based authentication (KERBEROS AS AN EXAMPLE)", *ITEE Journal: Inform. Technol. and Elec. Engin.*, vol. 5, no. 3, pp. 13–18, 2016.

[18] A. K. Jain, A. Ross, and S. Prabhakar, "An introduction to biometric recognition", *IEEE Trans. on Circ. and Syst. for Video Technol.*, vol. 14, no. 1, pp. 4–20, 2004.

[19] P. S. Teh, A. B. J. Teoh, and S. Yue, "A survey of Keystroke dynamics biometrics", *The Scientific World Journal*, vol. 2013, Article ID 408280, p. 24, 2013.

[20] S. P. Banerjee and D. Woodard, "Biometric authentication and identification using Keystroke dynamics: A Survey", *J. of Pattern Recogn. Res.*, vol. 7, no. 1, pp. 116–139, 2012.

[21] P. Dholi and K. P. Chaudhari, "Typing pattern recognition using Keystroke dynamics", in *Mobile Commun. and Power Engin.*, vol. 296, pp. 275–280, 2013.

Khaled Walid Mahmoud received a B.Sc. in Computer Science from Jordan University in 1992, a M.Sc. in Computer Science (Artificial Intelligence) from Jordan University in 1998 and a Ph.D. in Print Security and Digital Watermarking from Loughborough University, UK, in 2004. This was followed by academic appointments at Zarqa Private University (Assistance Professor in Computer Science). His areas of interest include information security, digital watermarking, image processing, AI and Arabic language processing.
E-mail: k.w.mahmoud@zu.edu.jo
Department of Computer Science
College of Information Technology
Zarqa University
P.O. Box 132222, Zarqa 13132, Jordan

Khalid Mansour received his Ph.D. in Computer Science from Swinburne University in 2014. He is currently the head of CS Department at Zarqa University, Jordan. His research interests include automated negotiation in multi-agent systems, web services and information security.

E-mail: kmansour@zu.edu.jo
Computer Science Department
Zarqa University
P.O. Box 132222, Zarqa 13132, Jordan

Alaa Makableh received her M.Sc. degree in Computer Science from Zarqa University in 2016 and a B.Sc. in Computer Information Systems from Hashemite University (Jordan) in 2007. Her research interest is in information security especially knowledge-based authentication systems.

E-mail: alaa.magableh@gmail.com
Computer Science Department
Zarqa University
P.O. Box 132222, Zarqa 13132, Jordan