sciendo

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **22**, No 2
Sofia • 2022 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2022-0013

A Scrutiny of Honeyword Generation Methods: Remarks on Strengths and Weaknesses Points

Yasser A. Yasser[1], *Ahmed T. Sadiq*[1], *Wasim AlHamdani*[2]

[1]*Computer Science Department, University of Technology, Iraq*
[2]*Information Technology Department, University of the Cumberlands, KY, USA*
E-mails: cs.19.28@grad.uotechnology.edu.iq Ahmed.T.Sadiq@uotechnology.edu.iq wasim.alhamdani@ucumberlands.edu

***Abstract*:** *Honeyword system is a successful password cracking detection system. Simply the honeywords are* (*False passwords*) *that are accompanied to the sugarword* (*Real password*). *Honeyword system aims to improve the security of hashed passwords by facilitating the detection of password cracking. The password database will have many honeywords for every user in the system. If the adversary uses a honeyword for login, a silent alert will indicate that the password database might be compromised. All previous studies present a few remarks on honeyword generation methods for max two preceding methods only. So, the need for one that lists all preceding researches with their weaknesses is shown. This work presents all generation methods then lists the strengths and weaknesses of 26 ones. In addition, it puts 32 remarks that highlight their strengths and weaknesses points. This research has proved that every honeyword generation method has many weaknesses points.*

***Keywords*:** *Flatness, honeychecker, honeywords, password, sweetwords.*

1. Introduction

Password-based User authentication has three factors, authentication by something the user knows (e.g., password), authentication by something the user has, and authentication by something the user is [1, 2]. Among these three authentication techniques, password-based authentication is widely accepted because of its simple login implementation and ease of memorability. Because of its popularity, password-based authentication schemes have also been explored using different attack models such as password cracking [3, 4]. Passwords are most commonly kept in a file system, or a database in plain text or hashed value format, mostly the password database listing is username/hashed password pairs. A password hash is a password once subjected to a one-way mathematical procedure or technique that results in an entirely new text [5, 6]. Password Cracking is the act of recovering passwords through an

unconventional and usually unethical method from data that has been stored or sent through a computer system [7, 8].

Honeywords is a simple method for improving the security of hashed passwords and facilitating the detection of password cracking. Several "honeywords" (Fake passwords) are related to every user's account [9, 10]. An attacker who has compromised the hashed passwords' database and then succeeds in reversing the hash function will not distinguish between the sugarword and the honeywords. The employment of a honeyword for the login process will trigger a "silent alarm" [11, 12]. Honeychecker is an auxiliary server that can distinguish the real password and triggers an alarm if a honeyword is entered. The honeychecker assumes that the connection with the login server has been via a secured channel that is authorized [13].

In 2013, J u l e s and R i v e s t [14] offered the honeyword system to reveal the login attempts using the hacked passwords. In 2014, E r g u l a r [15] has presented a few remarks to draw attention to potential flaws in the honeywords system of Jules in [14]. For all honeyword generation methods, since the appearance of the honeyword system [14] until the preparation of this study, the studies always have suggested a new generation method and presented a few remarks on one or two of previous methods only. Here the need for a study that lists all previous research and presents their strengths and weaknesses that has appeared.

This work presents different methodologies and techniques that generate honeywords, then lists the strengths and weaknesses for each method, 26 generation methods are listed, and 32 remarks that highlight the strengths and weaknesses points are presented. This research proves that there are many weaknesses in the honeyword generation methods, and every one has its weaknesses points.

Some Terminologies, Sweetwords: the sugarword and honeywords (k). Honeywords: the false passwords generated by the honeyword generation algorithm (k–1). Sugarword: the real password supplied by the user. Honeypot: fake and legitimate accounts, the fake is set up by the administrator to detect the breaches [16, 17].

2. Honeywords

The idea of the honeywords system is simple but brilliant. It depends on the generation of honeywords (False passwords) from the sugarword (Real password) and inserts them together into the user's account as sweetwords then hashes them all [18-20]. If the adversary succeeds in getting the plain passwords from the hashed passwords, he/she should make a right guess for the real password among the sweetwords. Otherwise, a silent alarm may be set off to the system's admin, indicating that password cracking may happen [21-23]. The procedure taken by the admins depends on the policy followed in the organization; it can be blocking, suspending, or warning the account [24, 25].

Flatness, let z be the attacker's possibility of successfully guessing the sugarword. This possibility is taken over the user selection of password pi, the generation technique Gen(k; pi), because an attacker can succeed with possibility 1/k

just by guessing sugarword at random [26-28]. Therefore, the honeyword generation process is considered perfectly flat. If the generation technique is as flat as possible (i.e., 1/*k* flat), the adversary has at least 1 – (1/*k*) chance of picking a honeywords [29, 30].

If the sweetwrods *k*=20, then for the perfectly flat honywords generation, the adversary has (1/20=5%) chance to pick the sugarword and (1–5%=95%) chance to pick a honeyword [31, 32].

User Login, When the user attempts login to his account, the login server checks the honeypot (Fake and legitimate accounts, the fake is set up by the administrator to detect the breaches) [33-35]. If his/her account is fakes then an alarm is sent to the administrative as a possible breach, else the account is legitimate then hashed the password of the user and compared to its database of sweetwords and sent (Check: *i*, *j*) to the honeychecker [36, 37].

The honeychecker keeps a single database value $c(i)$ for every user u_i (the $c(i)$ is the sugarword index); the values are tiny integers in the range of 1 to *k*, for some tines integer parameter *k* (e.g., *k*=20). The honeychecker takes just two sorts of commands:

- Set: *i*, *j*
  Sets $c(i)$ to have value *j*
- Check: *i*, *j*
  Checks that $c(i) = j$. Check results may be returned to the asking computer system. If the check fails, a "silent alarm" may be triggered [38, 39].

3. Honeyword generation methods

This section explains the generation methods of creating honeywords according to their date of publication.

3.1. In 2013

Juels and Rivest [14] split the honeyword generation methods depending on whether or not there is an effect on the User Interface (UI).

3.1.1. Legacy-UI based honeyword generation method

The legacy-UI does not inform the user about his/her usage of honeywords. At the same time, it does not interact with him to influence his password choice. Chaffing occurs when the password pi is chosen, and the honeyword generating technique Gen (*k*; pi) or "chaff procedure" creates a set of *k* – 1 extra different honeywords ("chaff").

A. Chaffing by tweaking

1) chaffing-by-tail-tweaking

Changing the tail of password characters at specific *t* locations.

For example, suppose the user's password is "TU-9g73", the list Wi for tail tweaking with *t* = 4 and *k* = 5 may be:

TU-2f45 TU-6h23 TU-0b12 TU-8j60 TU-5l63.

2) chaffing-by-tweaking-digits.

The last *t* locations with digits are selected. (If the password has lower than *t* digits, non-digit places can be used as needed [40]).

For example, if (*t* = 3): 762@jupiter 934@jupiter 815@jupiter

B. Chaffing-with-a-password-model

1) Simple model

Using a probabilistic model of actual passwords, this model might be based on a provided published list *L* of hundreds, maybe thousands of passwords, as well as maybe some other factors [41].

For example, iloveyou monkey sunshine

2) Modeling syntax

Honeywords are created using the same syntax as passwords. First, the password is decided into a series of "tokens". Every token represents a separate syntactic component such as an script, digit, or collection of symbols.

The password super90man, for example, will have tokens W5 |D2 |W3. Honeywords are then formed by substituting tokens with randomly picked values corresponding to the tokens moons64hat [42].

C. Chaffing with "tough nuts".

The system may also need honeywords that are far more difficult to break than the ordinary and so strong that an adversary would almost definitely never break them. (These "honeywords" may not even be passwords; they may simply be lengthy, e.g., 256-bit random bit strings).

For example, what should the attacker do with the list below?

princess asdfghjkl ? dragon qwertyuiop ?

3.1.2. Modified-UI based honeyword generation method

The UI informs the user about the honeywords usage or interacts with the user to influence him/her which password to choose.

A. Take-a-tail

The take-a-tail approach is similar to the chaffing-by-tail-tweaking approach. However, this approach is different in tail choosing; the tail of the new password here is picked at random by the system and needed in the user-entering a new password.

For example,

Make a password suggestion: ********

To create your new password, add "942" to the end.

Please enter your new password: **********

B. Random pick

This method asks the user to supply many passwords then the system arbitrarily choses one of them as a sugarword and informs the user about it, while the rest of the passwords are considered as honeywords.

3.1.3. Hybrid generation methods

By combining the advantages of several honeyword generating processes, a "hybrid" system may be created. This approach creates a hybrid legacy-UI technique by combining the Chaffing-with-a-password-model with a user-supplied password *p* to build a collection of ($a \geq 2$) seed sweetwords.

To create ($b \geq 2$) tweaks, apply chaffing-by-tweaking-digits to each seed sweetword (including the seed sweetword itself). This produces a complete set W of $k = a \times b$ sweetwords.

For example, if a=3 and b=3, then k=9.

soccer834	million934	dragon269
soccer248	million624	dragon184
soccer160	million052	dragon938

3.2. In 2014

Erguler [43] suggests an alternative approach for the honeyword generation that selects the honeywords from existing user passwords in the system to provide realistic honeywords. This method is called "Storage-index". The suggested method continues to rely on honeywords to identify password cracking. However, rather than generating honeywords and storing them in a password file, this approach advises using existing passwords to mimic honeywords. To do this, for each account, $k - 1$ existing password indexes, known as *honeyindexes*, are randomly allocated to a newly generated u_i account, where $k \leq 2$.

Furthermore, a random index number is assigned to this account, and a hash of the real password is stored in a list with a proper index. In a different list, u_i is recorded with an integer value set consisting of honeyindexes and proper index. Therefore, if an attacker compares both lists, he/she notices that every username is coupled with k integers as sweetindexes, every one of the indexes leads to correct passwords.

This method employs two files of passwords, F_1 and F_2, on the login server: As indicated in Table 1, F_1 saves the username and honeyindex set, $\langle hu_i; X_i \rangle$ pairings, where hu_i indicates a honeypot account. It is worth noting that each entry contains two items. The first part is the accounts' username, and the second is the honeyindex that has been established for the specific user. Furthermore, the table is arranged alphabetically by username column. F_2, otherwise, saves index number and password hash, $\langle c_i; H(p_i) \rangle$, as shown in Table 2. In this instance, every table entry has two items. The first item is the accounts' *sugarindex*, while the second is the hash of the related password.

The extra server honeychecker is used in this technique to keep valid indexes for every account and assumes that it is connected to the login server via a secure connection in an authorized way.

Table 1. F_1 Password file example for the suggested method [43]

Username	Honeyindex set
roza-marta	(76, 13459, … , 20645)
suarez	(57342, 98645, … , 99738)
mamamia20	(43, 2438, … , 67861)
:	:
pepsi-7	(675, 104256, … , 19854)
soocer90	(789, 14256, … , 45321)

Table 2. F_2 Password File Example for the Suggested method [43]

S_I	S_H
6	$H(p_6)$
43	$H(p_{43})$
77	$H(p_{77})$
:	:
220000	$H(p_{220000})$
220005	$H(p_{220005})$

3.3. In 2015

Chakraborty and Mondal [44] suggested new honeyword generation methods, which are: modified-tails, Close-Number-Formation (CNF), and caps-key.

3.3.1. Modified-tail generation method

As a tail for user password, the "modified-tail" approach asks the user to pick $m - 1$ items from a list of m of special characters S = [@,?,|]. By combining all of these special characters, other $m - 1$ honeyword will be generated.

For example, the user chooses the password "coffee" and tail "?|" from S, the user's password will be "coffee?|". The system will generate the following sweetwords:

coffee@?| coffee?@| coffee|?@
coffee|@? **coffee?|@** coffee@|?

The method appends the character that completes the set S (in the example case, "@") to the password given by the user. For the login process, when the user enters the password "coffee?|", the system then searches for the password "coffee?|@)". If the real password is found in sweetwords, the system sends its position to the honeychecker to ensure that the entered password is correct.

3.3.2. Close-Number-Formation (CNF) generation method

CNF method proposes to handle the honeyword generation process of the passwords that incorporates users' birthday or another meaningful date as part of their password. As a first step, it uses the number in the user password as a seed. Another input parameter is the total number of "honeywords" $k - 1$ to be created. Then the user will select the upper bound (should be efficient for passwords, date in a month generally not to override 30/31) of created "honeywords". Although, user has fixed it the upper bound is voluntary (if the user selects a day from the start or any other day of a month, then subsequent dates formed by CNF should not override 30/31). When the user selects an upper bound, the system considers two factors.

• The upper bound is selected by the user. It has to be larger than or similar to the number that exists in the password.

• The user-specified upper bound must let the system produce k "sweetwords" following the system's policy.

The user interface utilizing the CNF technique is illustrated below. The user interface's position field represents the number's location in the user password,

whereby the user is setting the upper bound. If the user has to pick more than one upper bound (or location), the user can do so by using "\".

The corresponding user's interface is seen below

user password selection: ******

Input Upper Bound: **

Position to enter: *

For example, the user picks 18 April 1988 as his real password and choose 19 as the upper bound, the resulting "sweetwords" using the CNF technique for $k = 4$ maybe

13April2002 17April1993 **18April1988** 12April2006.

3.3.3. Caps-Key Based generation method

Humans have a natural inclination to select passwords that contain lower-case characters. As a result, most authentication systems based on passwords require users to select a password with a minimum of eight characters. At the same time, the suggested "caps-key" technique chooses six characters as a minimum password length.

When a user wants to register himself/herself on a website, the system permits six characters as a minimum password length. Then user should choose two upper case characters for the password.

The corresponding user interface is seen below.

user password selection: ********

Warning: The password must include two uppercase letters

Password confirmation: ********

For example, if the user already meant to use "monkey" as his password, the user may utilize this process to pick "mONkey" as his real password, "sweetwords" for the user password as "mONkey" and $k = 4$ moNKey monKEy **mONkey** monkEY.

3.4. In 2017

Akshaya and Dhanabal [45] proposed a method that accepts graphical passwords (pictures) by using a chain from the image textual to form it like a password and storing it with a collection of unrealistic honeywords. As a result, an attacker who hijacks the hashed password database will be unable to recognize the correct password from the honeywords for any of the accounts because all passwords appear unreal.

The system asks the user to enter a picture as his password instead of entering the characters. After the reception of the picture, the system converts it into an alphanumeric chain. The system administrator will regard the first adjacent collection of characters from the chain as the user's password, while the remainder of the chain (names it as candyword) is stored on a different server (honeystore). The correct password chain is hashed and stored alongside a collection of randomly generated chains of the same length as honeywords. The correct password is saved in a separate list, and honeychecker stores the index of the correct password with its matching user id.

As a login process, there are two authentication processes, first when honeychecker confirms that the password is sugarword and the second when the honeystore appends sugarword to its matching candyword saved in the honeystore.

3.5. In 2017

Chakraborty and Mondal [46] suggested a new honeyword generating methodology based on a changed user interface, which is called Paired Distance Protocol (PDP). For login purposes, three pieces of information are required: a username, a password, and a password tail. The approach allows the user to choose his own password tail. The user picks a password tail of t >1 from a list of alphabet letters (a-z) and digits (0-9) during registration, together with the username and password. The default value of t in this scenario is deemed to be two. The characters from the set are spread in a circular list at random and in no particular sequence. As this leads to the creation of honeywords, this circular list is known as the Honey Circular List (HCL). In password file F_p, a system just saves one HCL. An instance of a HCL is shown in Fig. 1.

Based on the password tail selected by the user, PDP determines the paired distance between the components of the specified password tail. Distance between two components: The *paired distance* between two components c_1 and c_2, designated as PDP (c_1, c_2), is the total number of cells to be traversed in HCL clockwise to arrive from c1 to c_2; where $c_1 \neq c_2$.

Let u_i be the chosen username, password, and password tail be adam, fred, and mo, in that order. Then the paired distance between d and f (or, PD(m, o)) may be calculated as 5 using the HCL in Fig. 1. To reduce the effectiveness of an inversion attack, the system keeps the following login information for each user in F_p: username, password, and the calculated paired distance from the password tail. Besides the username, the honeychecker stores the first character (component) of the specified password tail. As a result, both F_P and honeyChecker are presented for the user interface as:

FP: adam fred 5

Honeychecker: adam m

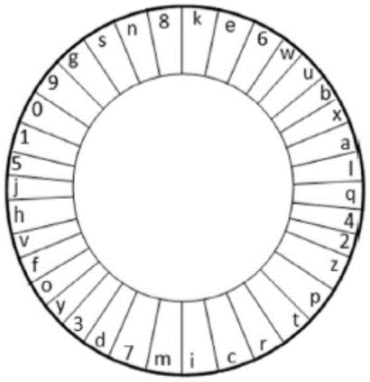

Fig. 1. HCL: Consist of alphabets and digits in random arrange [46]

3.6. In 2017

C h o r et al. [47] proposed a system for creating and storing honeywords in the honeypot, with honeywords generated from user information. For this, if an unauthorized person (stolen user's mobile) attempts to predict the password and the guessed password matches the honeywords, an alarm will be produced for the legitimate user, and only a login failure message will be sent to that user.

If the adversary wants to access the account, the system checks for honeywords in the system database and sends an email alert message to the valid user. The system also gives the MAC address and IP address of that system and temporarily blocks account. If the valid user wants to use his account, he enters it by hitting the link that got on the mail. The proposed system uses two-generation methods to generate the honeywords.

3.6.1. Personal details generation method

Honeywords are generated from the user details provided during the registration process of his banking account.

3.6.2. Existing user passwords generation method

The honeyword generation method chooses honeywords from existing user passwords in system administration to produce realistic honeywords and a fully flat honeyword creation technique.

3.7. In 2018

A k s h i m a et al. [48] suggested The "evolving-password model", the "user-profile model", and the "append-secret model" as enhanced and more functional honeyword generating methods.

3.7.1. Evolving-password model

The entire operation may be handled by two separate computation phases, which are mentioned below.

- Calculating the frequency of password patterns and tokens.
- Generating honeywords using precomputed frequencies and updating frequency lists, i.e., developing frequency lists each time the user registers a new password. Now, let's show how to generate honeywords from a given password: "wxyz789#". To construct honeywords, calculate the frequency of the pattern wxyz 789 # and the frequency of the tokens "wxyz", "789", and "#". Next, pick tokens with frequencies that are comparable to those of "wxyz", "789", and "#". The token "789" correspond to "5", "#" to "$", and "wxyz" to "code". As a result, one of the honeywords is "code$5".

3.7.2. User-profile model

This model produces honeywords by merging various user profile information and checking the minimal distance between the honeyword and the real password. A method for generating honeywords is to construct distinct sets from specified user information that includes tokens of each sort, such as "alphabet-strings",

"digit-strings", and "special-character-strings". Next, construct potential mixing of items from each token set. Then resulting items are used to construct honeywords. Following the instructions below is one method for making honeywords.

For example, suppose the following user profile information to be known:

Name: alex tony; Date of Birth: 03/03/2000; Address: 87 north 40 road; Name of the first teacher: smith and Password: adam$87road

For this user, the system can then create the following:

Digit tokens= 03, 03, 2000, 87, 40

Alphabet tokens= alex, tony, north, road, smith

SpecialChar tokens= / , $

Then Honeywords are: tony/2000 alex$03 smith#tony alex@40road.

3.7.3. Append-secret model

In this approach, the user enters his or her password during the registration process, then the system requests an additional entry, suppose *e* that can be 2 to 4 entries long to create a random string *s* with a length of 3 as the default, taking into account numbers, characters, and symbols. The model performs $f(p \| e \| s)$ and returns *r*, where *f* is a collision-resistance one-way function. Because *s* is chosen at random for every site, the intersection of information from several sites does not expose the real password, even if the user uses the same *e* for several sites.

For example, the user enters password: wxyz then enter the string of length (two-four): 2000.

System creates secret: &5n

System calculates: f(wxyz || 2000 || &5n) = 3j9t#

Database stores: H(wxyz || 3j9t#).

3.8. In 2018

Chakraborty, Singh and Mondal [49] proposed a questionnaire-based authentication method trying to generate perfect flat honeywords. In the flowchart of the method being proposed, given that recognition is significantly simpler than memory, the questionnaire-based method has the obvious benefit of allowing a user to identify the correct answer rather than remembering it. Fig. 2 depicts the fundamental aspects of the proposed questionnaire-based authentication approach.

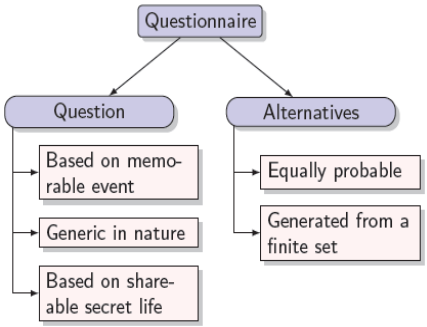

Fig. 2. The flowchart of the proposed questionnaire-based authentication method [49]

To generate sweetwords, while answering a questionnaire, a user needs to enter the index of the option and then give the correct answer. Each question has four options, ranging from A to D.

For example, what is the name of your first girlfriend? Recognize the first character,

(a) A (b) R (c) N (d) S.

As a result, if a user answers *s* questions, a response string of length *s* will be produced, comprising of characters ranging from A to D.

For example, if $s = 5$, suppose AADBC is the proper string. Here, if $n = 5$, the method keeps additional four alternative answer strings as honeywords. Below is the list of possible provided sweetwords for $n=5$:

ABCAD ABBCA CADCC **AADBC** ABABC,

where AADBC denotes the real password, and the rest words are the honeywords.

3.9. In 2019

A k i f et al. [50] suggested an alternative method that generates honeywords by using four techniques. As a consequence, four sets of honeywords are introduced to the system that seems like actual passwords.

3.9.1. Generate honeywords from existing user information

Building a database comprising public personal questions (fifty-sixty questions) separated into two sections based on replies. The first section is about the names, which will be turned into characters (childhood name, preferred country, preferred club, pet's name, or any similar questions). The second section will be about digits (birthday date, anniversary, the best year in your study, or any similar questions). Six questions from the database will be picked at random (three from each section). Then, by merging the first and second section answers, five honeywords will be formed. If a user does not wish to answer a question right away, the user might disregard it. Furthermore, if the real password only has two digits, the method will choose the digits from the digit's answers for the honeywords.

For example,

a) Characters section

Nickname? Junior Childhood name? Jojo Country? England

b) Digits section

Best year in your study? 2016 Wife birthday? 1988 In which year did you have surgery? 2002

The honeyword results will be:

Junior2016 England1988 Jojo2016 Junior1988 England2002.

3.9.2. Generate honeywords from a dictionary attack

This kind of honeyword is produced via a dictionary attack, with four formed in that sort of group. The basic idea behind creating appropriate honeywords is to utilize the original password with a change of up to three digits or characters after scanning through the dictionary attack. Some passwords are ineligible for this sort of group

because they are very hard to uncover in a dictionary attack. In this instance, from the other groups, four honeywords will be produced.

3.9.3. Generate honeywords from a generic password list

This honeyword group is based on the five hundred worst passwords list, with five honeywords picked randomly.

3.9.4. Generate honeyword form shuffling the characters

This form of honeyword is created by shuffling and then mixing in certain letters or numbers from the ID user. First, the original password with certain digits and characters is formed to be entered into the honeywords, followed by the generation of meaningless words. This stage involves the creation of 10 honeywords. The four sets of honeywords are then grouped with the real password to form 25 sweetwords.

3.10. In 2020

Fauzi, Yang and Martiri [51] proposed the PassGAN-based honeyword generation method trained on two types of datasets. As a result, three generation methods have been suggested. PassGAN is trained on a published password dataset, PassGAN has been trained on random password database and hybrid PassGAN-based techniques that combine the benefit of both.

PassGAN has been created to guess passwords [52]. PassGAN completes its mission with the help of a Generative Adversarial Network (GAN). GAN is a deep learning methodology capable of learning and producing unreal data comparable to its training data [53]. A GAN is often made up of two deep neural networks that compete each other: a generative model G and a discriminative model D are produced. G is given the responsibility of learning from the data that it has been trained on and creating some fresh samples that are similar to the data that has been given, whereas D is given the responsibility of determining whether every sample comes from the real training data or has been made by G. PassGAN, like the generic GAN, contains a generator model, which is trained based on actual password dataset (e.g., published password dataset) to generate unreal passwords and a discriminator model, which is trained to distinguish between the actual and fake passwords. PassGAN is built with "Improved Training of Wasserstein GANs" (IWGAN) [54] and optimized with ADAM [55].

The honeywords generating process of the proposed method uses the PassGAN. The database is employed to train the PassGAN, and once trained, the PassGAN's generator model is used to produce $k - 1$ of honeywords for each real password. This experiment employs three PassGAN-based techniques, which are as follows:

3.10.1. PassGAN trained on the published password database

The PassGAN employed to create honeywords is initially trained on a published password dataset. Because most of the passwords in the published data are simple to guess passwords, the created honeywords are also simple to guess passwords. This type of password is quite similar to passwords produced by people. However, this technique assumes that the adversary does not know that all of the original passwords

in the dataset have been produced by a machine. Therefore, the honeywords created by the PassGAN that has been trained on published data are designed to appear human-choice to deceive the attacker into selecting the correct honeyword.

3.10.2. PassGAN trained on the computer-generated password database

This technique assumes that the adversary is aware that all of the original passwords in the dataset have been produced by a computer. As a result, the technique requires honeywords to be developed that are identical to the original passwords. Furthermore, the PassGAN has been trained on a dataset of computer-generated passwords. Thus, the generated honeywords resemble computer-generated passwords.

3.10.3. Hybrid PassGAN-based

This technique combines the two preceding tactics to make the honeywords appear like a mix of computer-generated and human-choice passwords. The system benefits from this hybrid strategy since it gives the attacker a tiny probability of guessing the real password regardless of whether the attacker knows that all original passwords are computer-generated or not.

4. Remarks on honeyword generation methods

After examining the honeyword generation methods, this section will present remarks to highlight the strong and weak points.

4.1. Shared remarks among several honeyword generation methods

This section lists nineteen remarks, then distributed over the generation methods in two tables. Table 3, collects remarks on general properties, flatness, security, and efficiency. Table 4, collects remarks on properties related to requiring additional information for the honeyword generation process. Each remark in this section can be a strength or a weakness.

Remark 1. Flatness

This is the probability that the adversary can succeed in guessing the correct password among the false passwords, (more details in Section 2). Under certain conditions, all approaches can reach $1/k$ perfect flatness. Satisfying some conditions to achieve perfect flatness is a weakness, while not needing to satisfy any conditions is a strength. The * refers to satisfy the condition, which means nothing makes the correct password distinct from the fake passwords.

Remark 2. DoS resistance

The weak DoS resistance implies that an attacker may submit a honeyword with a high likelihood given knowledge of the password. The strong DoS resistance implies that a DoS attack is unlikely. Thus, weak resistance is a weakness, while strong is a strength.

Remark 3. MSV crisis

Multiple System Vulnerability crisis happens when the same password is used in two or more distinct systems that use the same honeyword generation method. If the systems are breached, an attacker can obtain the correct password of the

corresponding user by intersecting the lists of sweetwords. Therefore, undergoing to MSV crisis is a weakness, while not submitting is a strength. For example,

First system: **sun35shine** sun74shine sun22shine sun96shine,
Second system: iloveyou **sun35shine** monkey windows.

Remark 4. Typo-safety

Refer to user typing mistake that hit honeyword and causes trigger of the alert by honeychecker server. This usually occurs when the user's real password is similar to a honeyword. Having Typo-safety is a strength, while the lack of it is a weakness.

For example, star1 star3 **star6** star7.

Remark 5. Published list (available database)

Some of the honeyword generation methods depend on lists or databases of real passwords in the honeyword generation process. Reliance on this technique is not a good idea; such a list may also be available to the adversary, who could use it to help identify honeywords thus guessing the correct password. Thus, the use of this technique is a weakness, while not using them is a strength.

Remark 6. Leets

The leets are the use of character replacement in ways that play on the similarity of their glyphs to mimic the characters in the password. As an example, c@r is leet corresponding to word car. Hence, the adversary will succeed in guessing the correct password. Therefore, undergoing to leets issue is a weakness, while not submitting is a strength.

Remark 7. Recognizable pattern

Many users prefer to choose passwords that have a well-known pattern. The randomly replacement-based honeyword method seems a weakness against such passwords since the content solidity of such passwords would be compromised, and the true password would become very obvious. As a result, an adversary has noticeably recognized the correct password. Therefore, undergoing to recognizable pattern issue is a weakness, while not submitting is a strength.

For example, bond007 james007 007bond 007007.

Remark 8. Consecutive numbers

Because users prefer rememberable number patterns, many choose to use consecutive digits in their passwords, such as “123” or “1234”. If the generation method changes these numbers by random numbers, an attacker can simply differentiate the real password from the corresponding honeywords by examining the consecutive numbers. Therefore, undergoing consecutive numbers issue is a weakness, while not submitting is a strength. For example, moon813 moon612 **moon123** moon763.

Remark 9. Special date

Some users tend to choose numbers concerning the birth date, anniversary, the best year in their study, or any of the similar dates to include in their password. For such passwords, when they change their numbers, the date digits will be substituted with the digits chosen at random. As a result, an attacker may readily recognize honeywords and retrieve the real password. Therefore, undergoing to special date issue is a weakness, while not submitting is a strength.

For example, john7560 john9421 **john1987**

Table 3. Remarks on general properties, flatness, security, and efficiency

No	Article (Year)	Method	Remark									
			Flatness	DoS resis-tance	MSV crisis	Typo safety	Published list (Ava-ilable database)	Leets	Recogni zable pattern	Consecutive numbers	Special date	Correlation
1	Juels and Rivest [14] (2013)	Chaffing-by-tail-tweaking	Perfect flat If *	Weak	Yes	No	No	Yes	Yes	Yes	Yes	Yes
		Chaffing-by –tweaking-digits	Perfect flat If *	Weak	Yes	No	No	No	Yes	Yes	Yes	Yes
		Simple model	Perfect flat If *	Strong	Yes	Yes	Yes	No	No	No	No	Yes
		Modeling syntax	Perfect Flat If *	Strong	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
		Chaffing with “tough nuts”	N/A	Strong	No	Yes	No	N/A	No	N/A	N/A	No
		Take-a-tail	Perfect flat unconditi-onally	Strong	Yes	Yes	No	No	No	No	No	No
		Random pick	Perfect flat If *	Strong	Yes	No	No	No	Yes	No	No	Yes
		Hybrid generation methods	Perfect flat If *	Strong	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
2	Erguler [43] (2014)	Storage-index	Perfect flat If *	Weak	Yes	Yes	No	No	No	No	No	Yes
3	Chakraborty and Mondal [44] (2015)	Modified-tail	Perfect flat If *	Weak	No	No	No	Yes	No	No	No	No
		CNF	Perfect flat If *	Weak	Yes	No	No	No	No	No	No	Yes
		Caps-Key Based	Perfect flat If *	Weak	No	No	No	No	Yes	No	No	Yes
4	Akshaya and Dhanabal [45] (2017)	Graphical passwords (image)	Perfect flat uncondi-tionally	Strong	Yes	Yes	No	N/A	No	N/A	N/A	No
5	Chakraborty and Mondal [46] (2017)	PDP	Perfect flat If *	Strong	Yes	No	No	No	Yes	Yes	No	No
6	Chor et al. [47] (2017)	Personal details method	Perfect Flat If *	Weak	Yes	Yes	No	Yes	Yes	Yes	No	Yes
		Existing user passwords method	Perfect flat If *	Weak	Yes	Yes	No	No	No	No	No	Yes
7	Akshima et al. [48] (2018)	Evolving password model	Perfect flat If *	Strong	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
		User-profile model	Perfect flat If *	Weak	Yes	Yes	No	Yes	Yes	Yes	No	Yes
		Append-secret model	Perfect Flat If *	Strong	No	Yes	No	No	Yes	Yes	No	No
8	Chakraborty, Singh and Mondal [49] (2018)	Questionnaire-based method	Perfect Flat If *	Strong	Yes	No	No	N/A	Yes	N/A	N/A	No
9	Akif et al. [50] (2019)	User information method	Perfect Flat If *	Weak	Yes	Yes	No	Yes	Yes	Yes	No	Yes
		Dictionary attack method	Perfect flat If *	Weak	Yes	Yes	No	Yes	Yes	No	No	Yes
		Generic password list method	Perfect flat If *	Strong	Yes	Yes	Yes	No	No	No	No	Yes
		Shuffling characters method	Perfect flat If *	Weak	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
10	Fauzi, Yang and Martiri [51] (2020)	PassGAN trained on published password database	Perfect flat If *	Strong	Yes	Yes	Yes	No	No	No	No	Yes

Table 4. Remarks on properties related to requiring additional information for the honeyword generation process.

No	Article (Year)	Method	Remark								
			Modified-UI	User Information security issue	Registry with extra detail	Additional login activity	Memory stress	Non User-Friendly	Untruth or rubbish answer	System-Interfe-rence	Storage overhead
1	Juels and Rivest [14] (2013)	Chaffing-by-tail-tweaking	No	No	No	No	Low	No	No	No	No
		Chaffing-by –tweaking-digits	No	No	No	No	Low	No	No	No	No
		Simple model	No	No	No	No	Low	No	No	No	No
		Modeling syntax	No	No	No	No	Low	No	No	No	No
		Chaffing with “tough nuts”	No	No	No	No	N/A	No	No	No	Yes
		Take-a-tail	Yes	No	Yes	No	High	Yes	No	Yes	No
		Random pick	Yes	No	Yes	No	High	Yes	Yes	Yes	No
		Hybrid generation methods	No	No	No	No	Low	No	No	No	No
2	Erguler [43] (2014)	Storage-index	No	No	No	No	Low	No	No	No	Yes
3	Chakraborty and Mondal [44] (2015)	Modified-tail	Yes	No	Yes	No	High	Yes	No	Yes	No
		CNF	Yes	No	Yes	No	Low	Yes	No	Yes	No
		Caps-Key Based	Yes	No	Yes	No	High	Yes	No	Yes	No
4	Akshaya and Dhanabal [45] (2017)	Graphical passwords (image)	Yes	No	Yes	Yes	High	Yes	No	Yes	Yes
5	Chakraborty and Mondal [46] (2017)	PDP	Yes	No	Yes	Yes	High	Yes	No	Yes	Yes
6	Chor et al. [47] (2017)	Personal details method	Yes	Yes	Yes	No	High	Yes	Yes	Yes	Yes
		Existing user passwords method	No	No	No	No	Low	No	No	No	Yes
7	Akshima et al. [48] (2018)	Evolving password model	No	No	No	No	Low	No	No	No	No
		User-profile model	Yes	Yes	Yes	No	High	Yes	Yes	Yes	Yes
		Append-secret model	Yes	No	Yes	Yes	High	Yes	No	Yes	No
8	Chakraborty, Singh and Mondal [49] (2018)	Questionnaire-based method	Yes	Yes	Yes	Yes	High	Yes	Yes	Yes	Yes
9	Akif et al. [50] (2019)	User information method	Yes	Yes	Yes	No	High	Yes	Yes	Yes	Yes
		Dictionary attack method	No	No	No	No	Low	No	No	No	No
		Generic password list method	No	No	No	No	Low	No	No	No	No
		Shuffling characters method	No	No	No	No	Low	No	No	No	No
10	Fauzi, Yang and Martiri [51] (2020)	PassGAN trained on published password database	No	No	No	No	Low	No	No	No	Yes
		PassGAN trained on computer-generated password database	No	No	No	No	Low	No	NO	No	Yes

Remark 10. Correlation

One of the concerns is the correlation between username and password. So, the correct password may simply be identified from honeywords. Thus, an attacker may quickly guess the password from the corresponding honeywords. Therefore, undergoing to correlation issue is a weakness, while not submitting is a strength.

For example, Username: mark Password: mark999.

Remark 11. Modified-UI

The UI informs the user somehow about the usage of honeywords, the UI interactions with the user for longer than just the username and password input. The

modified-UI requires extra activities from users, reducing usability and making it the user's least favorite UI. Thus, using Modified-UI in the system is a weakness, while using legacy-UI is a strength.

Remark 12. User information security issue

Several honeyword generation methods use a technique that leans on personal knowledge-based questions, forcing the user to provide personal information and detail, to help the methods to generate honeywords. If the system is compromised and personal information disclosed, this information may be used on another system and threaten the user. Thus, using this technique constitutes a security issue considering it a weakness, while not using it is a strength.

Remark 13. Registry with extra detail

The registration process of some honeyword generation methods imposes the user to provide extra detail beyond the username and password; obviously, that comprise remembering extra information. However, these methods are often not preferred by the user. Therefore, a registry with extra detail is a weakness, while a registry with only a username and password is a strength.

Remark 14. Additional login activity

Some of the generation methods enforce the user to perform additional login activity by implementing action and activity exceeding the submission of username and password. These methods in most situations are not preferred by the user. Therefore, logging in with extra detail is a weakness, while login in with only username and password is a strength.

Remark 15. Memory stress

High stress on memory happens when the system asks the user to extra memorize something inconsequential or irrelevant to the user. Such systems are burdensome for the user and may lead to wrong entry, so they are not preferred for the user. Thus, high stress on memory is a weakness, while low stress is a strength.

Remark 16. Non-User-friendly

The method is non-user-friendly if it is using a technique that forces the user for extra memorizing, generally used for generating the honeywords. The user does not prefer such systems because such systems are burdensome. Thus, using this technique is a weakness, while not using it is a strength.

Remark 17. Untruth or rubbish answer

Some of the honeyword generation methods use a technique that relies on personal knowledge-based questions, especially the questions that have the character of privacy, in most cases, users do not answer truthfully. In another case, the user may get bored with the questions then provide a rubbish answer. In the two cases, the system is considered to be an unsecured system. Thus, using this technique is a weakness, while not using it is a strength.

Remark 18. System interference

Refers to the system using a technique that influences the password that users choose. The user does not prefer such systems because the password will be hard to remember therefore exposed to errors. Thus, using this technique is a weakness, while not using it is a strength.

Remark 19. Storage overhead

The storage costs assume store *k* of sweetwords, some systems requiring extra storage costs considered storage overhead. Requiring extra storage cost is a weakness, while the opposite is a strength.

4.2. Custom remarks each for a single honeyword generation method

This section is listing thirteen remarks, each one custom for a specific generation method. All remarks in this section consider weaknesses.

Remark 20. Meaningful word issue

On: Chaffing-by-tail-tweaking method [14]. If the user prefers to append a meaningful word to a password, tweaking for letters in the word will change it to a non-meaningful word. Thus, the adversary can easily guess the correct password.

For example, **57*flavors** 57*flavrbn 57*flaavctz.

Remark 21. Rubbish word issue

On: Modeling syntax method [14], the rubbish word that is not present in the dictionary weakens the results of these methods because such a word does not blend in with generated honeywords.

Remark 22. Tough nuts issue

On: Chaffing with "tough nuts" method [14], the attacker may assume that most passwords consisting of easy character and number combinations exclude the idea of the tough nut. Thus, the adversary will launch an attack while excluding the tough nuts, contrary to method assumptions.

Remark 23. Resetting password issue

On: Take-a-tail method [14], to get a preferred tail, the user may attempt to reset the password several times. Thus, the flatness feature is weakened.

Remark 24. Mistakenly submit a honeyword

On: Random pick method [14], the user may recall and accidentally submit a sweetword previously given and used by the system as a honeyword. As a result, the alarm will be triggered, warning of a possible breach.

Remark 25. DoS resistance issue

On: Storage-index method [43], if an attacker makes a large number of users accounts with the same password, the password is likely to seem like a honeyword for the actual accounts. Thus, the adversary's chance of realizing a DoS attack increases.

Remark 26. Upper case letters position

On: Caps-Key Based generation method [44], the position of upper-case letters should not indicate separate words or follow a specific pattern like the first two positions. Thus, the adversary can perform the right guess for the correct password. For example, MOnkey MonkeY MonkeyBanana

Remark 27. Image availability issue

On: Graphical passwords method [45], if the image password is not available because of mistakenly deleted, the device is damaged or stolen. That means a loss of passwords.

Remark 28. Storing and sending image issue

On: Graphical passwords method [45], the textual form of the image password may change by store or send an image. As a result, the string of image will not match the string of correct passwords saved in the system. Therefore, the login process will fail.

Remark 29. Choosing a tail issue

On: PDP method [46], this method did not provide for perfect flatness unless the user chooses a high random tail.

Remark 30. Flat alternatives issue

On: Questionnaire-based method [49], each question in the system should satisfy three properties, and each alternative should satisfy two properties in order to provide flat alternatives. Otherwise, the adversary may choose to make the right answers for the questions and guess the correct password.

Remark 31. Flat alternatives issue

On: Questionnaire-based method [49], give answers in a certain pattern, such as choosing the first, last, or consecutive alternatives. Thus, the adversary can perform the right guess for the correct password. For example, AAAAA ABCDD

Remark 32. Choosing a tail issue

On: Dictionary attack method [50], this method does not always generate honeywords because some passwords cannot be found in a dictionary attack.

5. Conclusion

This research introduces an examination with a simple explanation of honeyword generation methods of ten articles that have 26 generation methods, from J u e l s and R i v e s t [14] in 2013 to F a u z i, Y a n g and M a r t i r i [51] in 2020. Furthermore, it presents 32 remarks on the 26 generation methods. The remarks highlight the strengths and weaknesses points, 19 remarks share several methods, and 13 remarks are custom each for a single method. The shared remarks are distributed over methods in Table 3 and Table 4. this research proves there are many weaknesses in all honeyword generation methods.

As analysis for Table 3 and Table 4 shows, there have been only two methods that have a perfect flatness unconditional property (Take-a-tail and Graphical passwords) as a strength, on the other hand, the two methods have a weakness in modified-UI, stress on memory, non-user-Friendly, MSV crisis, and additional login activity. The methods that have a strong DoS resistance property as a strength, on the other hand, almost have a weakness in MSV crisis, leets, recognizable pattern, consecutive numbers, special date, correlation. The methods that have no MSV crisis property as a strength, on the other hand, almost have a weakness in conditionally perfect flatness, modified-UI, registry with extra detail, stress on memory, non-user-friendly, and system inference. The methods that have a typo-safety property as a strength, on the other hand, almost have a weakness in MSV crisis and correlation.

This study suggests recommendations for honeyword generation methods by taking advantage of remarks to avoid weakness points (especially the remarks in Table 3). Furthermore, it suggests combining many methods to bypass the largest

possible number of weaknesses. Further research in this field could be directed towards present remarks on other aspects of the honeyword technique other than the methods of generation, such as the policies against the potential breach or the applications where honeywords have been used.

References

1. Mohammed, A. A., A. K. Abdul-Hassan, B. S. Mahdi. Authentication System Based on Hand Writing Recognition. – In: Proc. of 2nd Scientific Conference of Computer Sciences (SCCS'19), March 2019, pp. 138-142. DOI: 10.1109/SCCS.2019.8852594.
2. Mukthineni, V., R. Mukthineni, O. Sharma, S. J. Narayanan. Face Authenticated Hand Gesture Based Human Computer Interaction for Desktops. – Cybernetics and Information Technologies, Vol. **20**, 2020, No 4, pp. 74-89.
3. Ahmed Tariq Sadiq, A. A. A., Sura Ali. Attacking Classical Cryptography Method Using Pso Based on Variable Neighborhood Search. – International Journal of Computer Engineering & Technology (IJCET), 2014.
   https://www.iaeme.com/ijcet.asp
4. Qasaimeh, M., R. S. Al-qassas, S. Aljawarneh. Recent Development in Smart Grid Authentication Approaches : A Systematic Literature Review. – Cybernetics and Information Technologies, Vol. **19**, 2019, No 1, pp. 27-52.
5. Alaa Kadhim, F., H. I. Mhaibes. A New Initial Authentication Scheme for Kerberos 5 Based on Biometric Data and Virtual Password. – In: Proc. of International Conference on Advanced Science and Engineering (ICOASE'18), 2018, pp. 280-285. DOI: 10.1109/ICOASE.2018.8548852.
6. Sadiq, A. T., L. Ali. Attacking Transposition Cipher Using Improved Cuckoo Search. – Journal of Advanced Computer Science and Technology Research, Vol. **4**, 2014, No 1, pp. 22-32.
   http://www.sign-ific-ance.co.uk/index.php/JACSTR/article/view/385
7. Chaudhari, S., R. Aparna, A. Rane. A Survey on Proxy Re-Signature Schemes for Translating One Type of Signature to Another. – Cybernetics and Information Technologies, Vol. **21**, 2021, No 3, pp. 24-49.
8. Abed, T. M., H. B. Abdul-Wahab. Anti-Phishing System Using Intelligent Techniques. – In: Proc. of 2nd Scientific Conference of Computer Sciences (SCCS'19), March 2019, pp. 44-50. DOI: 10.1109/SCCS.2019.8852601.
9. Genç, Z. A., S. Kardaş, M. S. Kiraz. Examination of a New Defense Mechanism: Honeywords. –Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. **10741**. G. P. Hancke, E. Damiani, Eds. Cham, Springer International Publishing, 2018, pp. 130-139.
10. Kute, S., V. Thite, S. Chopade. Achieving Security using Honeyword. – Int. J. Comput. Appl., Vol. **180**, Jun 2018, No 49, pp. 43-47. DOI: 10.5120/ijca2018917333.
11. Win, T., K. S. M. Moe. Protecting Private Data Using Improved Honey Encryption and Honeywords Generation Algorithm. – Adv. Sci. Technol. Eng. Syst., Vol. **3**, 2018, No 5, pp. 311-320. DOI: 10.25046/aj030537.
12. Chakraborty, N., S. Mondal. Towards Improving Storage Cost and Security Features of Honeyword Based Approaches. – Procedia Comput. Sci., Vol. **93**, 2016, No September, pp. 799-807. DOI: 10.1016/j.procs.2016.07.298.
13. Kusuma, A. B., Y. R. Pramadi. Implementation of Honeywords as a Codeigniter Library for a Solution to Password-Cracking Detection. – In: Proc. of IOP Conf. Ser. Mater. Sci. Eng., Vol. **508**, May 2019, No 1, 012134. DOI: 10.1088/1757-899X/508/1/012134.
14. Juels, A., R. L. Rivest. Honeywords: Making Password-Cracking Detectable. – In: Proc. of 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS'13), 2013, No October 2015, pp. 145-160. DOI: 10.1145/2508859.2516671.
15. Erguler, I. Some Remarks on Honeyword Based Password-Cracking Detection. – IACR Cryptol. ePrint Arch., Vol. **2014**, 2014, 323.
   https://eprint.iacr.org/2014/323.pdf

16. T h a k u r, P. V. Honeywords: The New Approach for Password Security. – Int. J. Res. Appl. Sci. Eng. Technol., Vol. **7**, April 2019, No 4, pp. 2449-2450. DOI: 10.22214/ijraset.2019.4446.
17. G h a r e, H. Securing System Using Honeyword and MAC Address. – Int. J. Res. Appl. Sci. Eng. Technol., Vol. **7**, May 2019, No 5, pp. 2685-2689. DOI: 10.22214/ijraset.2019.5446.
18. W a n g, R., H. C h e n, J. S u n. Phoney: Protecting Password Hashes with Threshold Cryptology and Honeywords. – Int. J. Embed. Syst., Vol. **8**, 2016, No 2-3, pp. 146-154. DOI: 10.1504/IJES.2016.076108.
19. P a l a n i a p p a n, S., V. P a r t h i p a n, S. S t e w a r t K i r u b a k a r a n, R. J o h n s o n. Secure User Authentication Using Honeywords. – Lecture Notes on Data Engineering and Communications Technologies, Vol. **31**, 2020, pp. 896-903.
20. S u r y a w a n s h i, B. D., P. B. T a y a d e, A. V. P a t i l, J. B. P a t i l, D. V. R a j p u t. Enhancing the Security Using Honeywords. – IJIRCT1601039 Int. J. Innov. Res. Creat. Technol., Vol. **208**, 2017, No 6, pp. 208-211.
www.ijirct.org
21. G u o, Y., Z. Z h a n g, Y. G u o. Superword: A Honeyword System for Achieving Higher Security Goals. – Comput. Secur., Vol. **103**, April 2021, 101689. DOI: 10.1016/j.cose.2019.101689.
22. L a n j u l k a r P r i t e e, I. V., I. R u p a l i, L. A r t i. Honeywords : A New Approach for Enhancing Security. – Int. Res. J. Eng. Technol., Vol. **06**, 2019, No 03, pp. 1360-1363.
https://www.irjet.net/archives/V6/i3/IRJET-V6I3256.pdf
23. S i v a j i, N., K. S. Y u v a r a j. Improving Usability of Password Management with Storage Optimized Honeyword Generation. – Int. J. Sci. Res. Sci. Technol., Vol. **4**, 2018, No 5, pp. 55-60. DOI: 10.32628/IJSRST184531.
24. P a g a r, V. R., R. G. P i s e. Strengthening Password Security through Honeyword and Honeyencryption Technique. – In: Proc. of Int. Conf. Trends Electron. Informatics, ICEI 2017, Vol. **2018-January**, 2018, pp. 827-831. DOI: 10.1109/ICOEI.2017.8300819.
25. H. R. B. B. S. J. Web Application: (with) HoneyWords and HoneyEncryption. – Int. J. Sci. Res., Vol. **4**, 2015, No 2, pp. 2313-2316.
https://www.ijsr.net/archive/v4i2/SUB151773.pdf
26. G e n ç, Z. A., G. L e n z i n i, P. Y. A. R y a n, I. V a z q u e z S a n d o v a l. A Critical Security Analysis of the Password-Based Authentication Honeywords System Under Code-Corruption Attack. –Communications in Computer and Information Science, Vol. **977**, 2019, pp. 125-151.
27. B r i n d t h a, J., K. R. H i t h a e i s h i n i, R. K o m a l a, G. A b i r a m i, U. A r u l. Identification and Detecting of Attacker in a Purchase Portal Using Honeywords. – In: Proc. of 3rd IEEE Int. Conf. Sci. Technol. Eng. Manag. (ICONSTEM'17), Vol. **2018-January**, 2017, pp. 389-393. DOI: 10.1109/ICONSTEM.2017.8261414.
28. B a m a n e, S. Achieving Flatness Using Honeywords Generation Algorithm. – Int. J. Res. Appl. Sci. Eng. Technol., Vol. **7**, May 2019, No 5, pp. 3491-3496. DOI: 10.22214/ijraset.2019.5572.
29. C a t u o g n o, L., A. C a s t i g l i o n e, F. P a l m i e r i. A Honeypot System with Honeyword-Driven Fake Interactive Sessions. – In: Proc. of Int. Conf. High Perform. Comput. Simulation (HPCS'15), 2015, pp. 187-194. DOI: 10.1109/HPCSim.2015.7237039.
30. F a u z i, M. A., B. Y a n g, E. M a r t i r i. Password Guessing-Based Legacy-UI Honeywords Generation Strategies for Achieving Flatness. – In: Proc. of 44th IEEE Annu. Comput. Software, Appl. Conf. (COMPSAC'20), 2020, pp. 1610-1615. DOI: 10.1109/COMPSAC48688.2020.00-25.
31. G a d g i l, M. A. A. Enhancing Security in User Authentication through Honeyword. – Int. J. Sci. Res. Manag., Vol. **4**, Jun 2016, No 6, pp. 4347-4350. DOI: 10.18535/ijsrm/v4i6.17.
32. N a t h e z h t h a, T., V. V a i d e h i. Honeyword with Salt-Chlorine Generator to Enhance Security of Cloud User Credentials. – Commun. Comput. Inf. Sci., Vol. **746**, 2017, pp. 159-169. DOI: 10.1007/978-981-10-6898-0_13.
33. M o e, K. S. M., T. W i n. Improved Hashing and Honey-Based Stronger Password Prevention against Brute Force Attack. – In: Proc. of International Symposium on Electronics and Smart Devices (ISESD'17), Vol. **2018-January**, October 2017, pp. 1-5. DOI: 10.1109/ISESD.2017.8253295.

34. S h a m i n i, P. B., E. D h i v y a, S. J a y a s r e e, M. P. L a k s h m i. Detection and Avoidance of Attacker Using Honey Words in Purchase Portal. – In: Proc. of 3rd International Conference on Science Technology Engineering & Management (ICONSTEM'17), Vol. **2018-January**, March 2017, pp. 260-263. DOI: 10.1109/ICONSTEM.2017.8261290.
35. W a n g, D., H. C h e n g, P. W a n g, J. Y a n, X. H u a n g. A Security Analysis of Honeywords. – In: Proc. of NDSS-Symposium, 2018, No February. DOI: 10.14722/ndss.2018.12345. **https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_02B-2_Wang_paper.pdf**
36. K a r t h i k, A., M. D. K a m a l e s h. Rat Trap: Inviting, Detection & Identification of Attacker Using Honey Words in Purchase Portal. – In: Proc. of 3rd International Conference on Science Technology Engineering & Management (ICONSTEM'17), Vol. **2018-January**, March 2017, pp. 130-132. DOI: 10.1109/ICONSTEM.2017.8261268.
37. J u e l s, A. A Bodyguard of Lies. – In: Proc. of 19th ACM Symposium on Access Control Models and Technologies (SACMAT'14), 2014, pp. 1-4. DOI: 10.1145/2613087.2613088.
38. S h i n d e, P. D., S. H. P a t i l. Secured Password Using Honeyword Encryption. – Iioab J., Vol. **9**, 2018, No 2, SI, pp. 78-82. **https://www.iioab.org/IIOABJ_9.2_78-82.pdf**
39. G e n ç, Z. A., G. L e n z i n i, P. Y. A. R y a n, I. V. S a n d o v a l. A Security Analysis, and a Fix, of a Code-Corrupted Honeywords System. – In: Proc. of 4th International Conference on Information Systems Security and Privacy, Vol. **2018-January**, 2018, No Icissp, pp. 83-95. DOI: 10.5220/0006609100830095.
40. Z h a n g, Y., F. M o n r o s e, M. K. R e i t e r. The Security of Modern Password Expiration. – In: Proc. of 17th ACM Conference on Computer and Communications Security (CCS'10), 2010, 176. DOI: 10.1145/1866307.1866328.
41. W e i r, M., S. A g g a r w a l, B. D e M e d e i r o s, B. G l o d e k. Password Cracking Using Probabilistic Context-Free Grammars. – In: Proc. of IEEE Symposium on Security and Privacy, May 2009, pp. 391-405. DOI: 10.1109/SP.2009.8.
42. B o j i n o v, H., E. B u r s z t e i n, X. B o y e n, D. B o n e h. Kamouflage: Loss-Resistant Password Management," – Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. **6345 LNCS**, 2010, pp. 286-302.
43. E r g u l e r, I. Achieving Flatness: Selecting the Honeywords from Existing User Passwords. – IEEE Trans. Dependable Secur. Comput., Vol. **13**, March 2015, No 2, pp. 284-295. DOI: 10.1109/TDSC.2015.2406707.
44. C h a k r a b o r t y, N., S. M o n d a l. Few Notes towards Making Honeyword System More Secure and Usable. – In: Proc. of Int. ACM Conf. Ser., Vol. **08-10-September**, 2015, No September 2015. DOI: 10.1145/2799979.2799992.
45. A k s h a y a, K., S. D h a n a b a l. Achieving Flatness from Non-Realistic Honeywords. – In: Proc. of International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS'17), March 2017, pp. 1-3. DOI: 10.1109/ICIIECS.2017.8276120.
46. C h a k r a b o r t y, N., S. M o n d a l. On Designing a Modified-UI Based Honeyword Generation Approach for Overcoming the Existing Limitations. – Comput. Secur., Vol. **66**, 2017, pp. 155-168. DOI: 10.1016/j.cose.2017.01.011.
47. C h o r, A., A. G a w a l i, A. M o h i t e, M. T a n p u r e, P. S. P. B., P. T. P. B. Improving Security Using Honeyword for Online Banking Authentication System. – IJARCCE, Vol. **6**, March 2017, No 3, pp. 976-978. DOI: 10.17148/IJARCCE.2017.63226.
48. A k s h i m a, A., D. C h a n g, A. G o e l, S. M i s h r a, S. K. S a n a d h y a. Generation of Secure and Reliable Honeywords, Preventing False Detection. – In: IEEE Trans. Dependable Secur. Comput. Vol. **5971**. No c. 2018, pp. 1-13. DOI: 10.1109/TDSC.2018.2824323.
49. C h a k r a b o r t y, N., S. S i n g h, S. M o n d a l. On Designing a Questionnaire Based Honeyword Generation Approach for Achieving Flatness. – In: Proc. of 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'18), August 2018, pp. 444-455. DOI: 10.1109/TrustCom/BigDataSE.2018.00071.

50. A k i f, O. Z., A. F. S a b e e h, G. J. R o d g e r s, H. S. A l-R a w e s h i d y. Achieving Flatness: Honeywords Generation Method for Passwords Based on User Behaviours. – Int. J. Adv. Comput. Sci. Appl., Vol. **10**, 2019, No 3, pp. 28-37. DOI: 10.14569/IJACSA.2019.0100305.
51. F a u z i, M. A., B. Y a n g, E. M a r t i r i. PassGAN Based Honeywords System for Machine-Generated Passwords Database. – In: Proc. of 6th IEEE Intl. Conf. Big Data Secur. Cloud, BigDataSecurity 2020, 2020 IEEE Intl. Conf. High Perform. Smart Comput. HPSC 2020 2020 IEEE Intl Conf. Intell. Data Secur. IDS 2020, pp. 214-220, 2020. DOI: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00046.
52. H i t a j, B., P. G a s t i, G. A t e n i e s e, F. P e r e z-C r u z. PassGAN: A Deep Learning Approach for Password Guessing. – Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. **11464 LNCS**, 2019, pp. 217-237. DOI: 10.1007/978-3-030-21568-2_11.
53. G o o d f e l l o w, I. et al. Generative Adversarial Networks. – Communications of the ACM, Vol. **63**, 2020, No 11. pp. 139-144. DOI: 10.1145/3422622.
54. G u l r a j a n i, I., F. A h m e d, M. A r j o v s k y, V. D u m o u l i n, A. Courville. Improved Training of Wasserstein GANs. – Advances in Neural Information Processing Systems, Vol. **2017-December**, 2017, pp. 5768-5778.
55. Q i, P., W. Z h o u, J. H a n. A Method for Stochastic L-BFGS Optimization. – In: Proc. of 2nd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA'17), 2017, pp. 156-160. DOI: 10.1109/ICCCBDA.2017.7951902.

Received: 25.01.2022; Second Version: 17.02.2022; Accepted: 25.02.2022 (fast track)