Who Are You? - a word embedding unlock method which defense peeking attack

En-Ming, Guo
CSIE, National Taiwan University
b05902068@ntu.edu.tw

Chun-Yu, Chen
CSIE, National Taiwan University
b05902058@ntu.edu.tw

Chin-Tang, Chen
CSIE, National Taiwan University
b05902061@ntu.edu.tw

Zih-Min, Wang
CSIE, National Taiwan University
b05902062@ntu.edu.tw

Hsiang-Sheng, Tsai
CSIE, National Taiwan University
b05902066@ntu.edu.tw

ABSTRACT

Today's smart devices unlock systems include password authentication, PIN code, gesture lock and face id - a facial recognition system recently announced by **Apple Inc**. Suppose there exists any malicious adversaries who have the ability to monitor or eavesdrop your device when you unlocking it, your personal sensitive data would exposed to severe danger.

For the purpose of solving this problem, we come up with a solution which takes good advantage of **word embedding**. When a user wants to register an account in our login system, he should enter an English word as password and post it with his own user name, the request would pass the system's authentication if the password exists in our vocabulary. After successful registration, the most important part of our design is showed in front of the user after trying to log in - password authentication. Our design includes 3 versions, which we will describe in detail later. In addition to introducing these three versions, we also enumerate the attack and defense results that these three versions may receive under our threat model, as well as their respective strengths and weaknesses.

ACM Reference Format:
En-Ming, Guo, Chun-Yu, Chen, Chin-Tang, Chen, Zih-Min, Wang, and Hsiang-Sheng, Tsai. 2019. Who Are You? - a word embedding unlock method which defense peeking attack. In *Taipei '19: CNS final, June 24–25, 2019, Taipei, TW*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the age of information, almost every one has their own mobile devices like cell phones and computers. The amount of personal information kept in those devices is tremendous and some of them may be very sensitive, for example, browsing history and messages sent between close friends. Thus, it is important to have a good method to protect the data.

The most commonly used approach today is password authentication. However, we think this method is very risky and can be susceptible to a very common shoulder-surfing attack, in which case a person tries to spy on the password a user types in and then replicates it afterwards. A extreme case of this attack would be typing password in a place with CCTV cameras, which is also very common today. Other kinds of unlocking method include the use of fingerprint and appearance of a person. However, these methods would require a special device to authenticate a user, thus making them restricted to only a certain usage, for example, they are commonly used to unlock a cell phone but it is rarely to see them being used to log in a website.

The main idea of our log-in method is to reveal as little information related to our password as possible during authentication. Every time, the password typed in would be different and, ideally, only makes sense to the valid user making an attacker eavesdropping the authentication process think the user is only making random choices.

The rest of this paper is structured as follows. First, in section 2 we define our problem formally. Then, section 3 contains the main structure of our log-in method. Also, we have tried 3 different designs of question scheme that would be introduced in section 4. The following is the analysis of our experiments on our system in section 5. Next, section 6 shows a brief introduction of related works. Finally, there are conclusions and future works in section 7, and references in section 8.

2 PROBLEM DEFINITION

2.1 Goals

In this paper, we will propose a unlocking method based on word embedding, a concept that calculates similarities between words, to achieve two main goals.

- `shoulder-surfing proof`: As described above, we want to be able to prevent shoulder-surfing attack while we are logging in a service.
- `memory-efficient`: The effort has to be made to use a log-in method is a top consideration for the design of it. Thus we want a user to memorize as little thing as possible when using our log-in method.

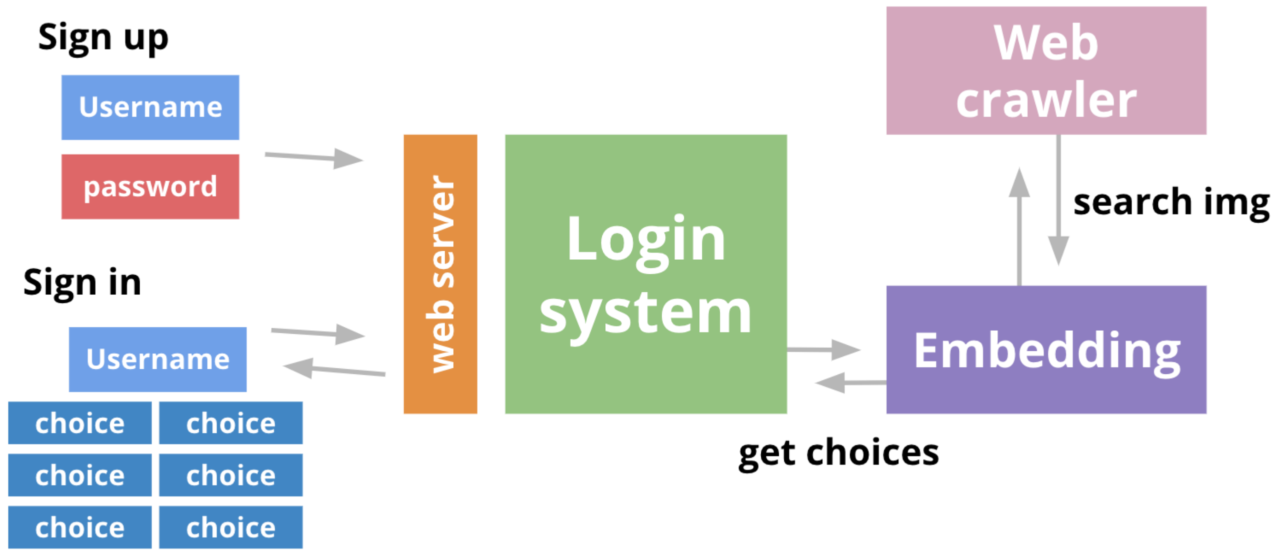


Figure 1: System Structure

2.2 Thread Model

In our system, identities can be divided into users, authentication servers and attackers. An user can successfully logs in the server by answering the server's question, and an attacker pretends to be the user to log in the server. We assume that the attackers are mainly divided into two categories: "the normal people who knows nothing" and "the people who hold the word embedding model and can peek at all users' answer when logging in". For "the people who doesn't know anything", they won't know what the user's previous registered password is and what he replied at the time of login, and they are completely unaware of the user's usual habits and preferences for setting a password. For an attacker who "holds the word embedding model and can peek at all users' answer when logging in", he won't know the user's previous registered password and the habit of using the password, but he can see the user's previous response to the server when one successfully logged in, and use his word embedding model to try to calculate the answer to the question asked by the server. Here we consider that the attacker's word embedding model are identical with ours, and even if the attacker has a slightly different word embedding model or maybe use the human knowledge for word embedding, it will only decrease the success rate in our system slightly. The difference could be ignored. Therefore, for the sake of simplicity, we only define these two types of attackers.

3 SYSTEM STRUCTURE

Our System is composed of three subsystem, including:

- `Embedding System`: handling the calculation of similarity and produce choices from embedding vector space.
- `Front-end User Interface`: handling user input and provide a graphic interface for user to use
- `Login Handler`: control the subsystems and make decision about whether a user is valid and allowed to login

We would explain in more detail as below.

3.1 Embedding Systems

3.1.1 model selection and preprocessing. Although there are many ways to calculate for the similarity and relevance between words, **word2vec** seems to be the best choice for us to implement our system. While projecting every word into high dimensional space, we could easily quantify the similarity as the cosine distance.

There are also lots of available pre-trained word2vec resource online, and finally we choose gensim's implementation and it's pre-trained model as our embedding vectors.

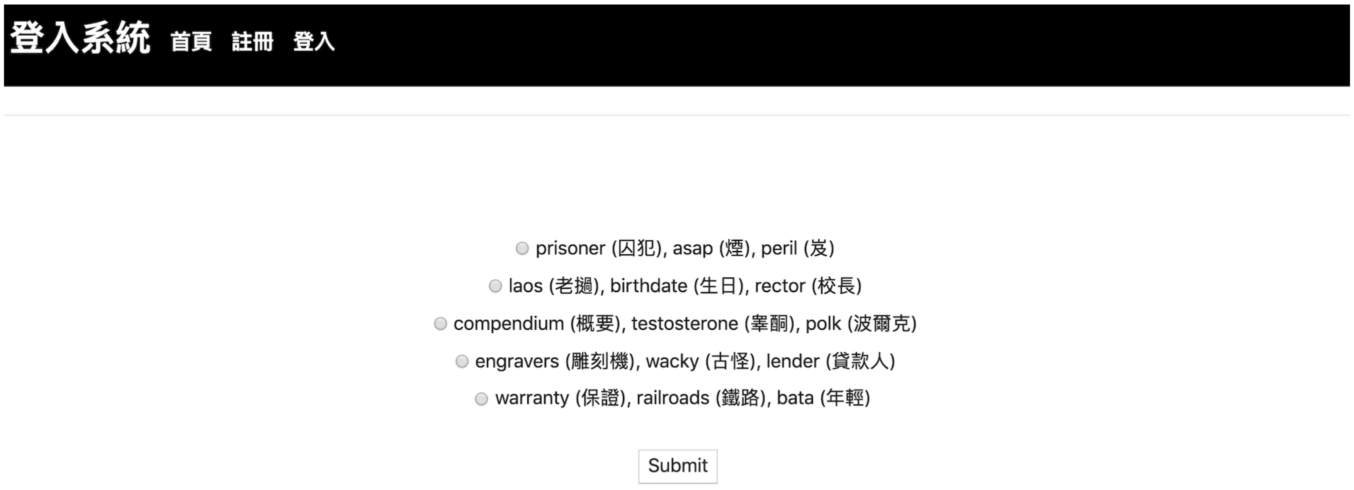
We also use some modification before deploying on our system, including:

- `Noun Filtering`: We choose terms which are mainly use as noun because it may present more concrete concept and easy for turning into image format. The other reason is to prevent the bias between the part-of-speech while counting similarity.
- `Vocabulary Trimmimg`: We choose the common words to be the candidates of our choose, because for those rare words, users could have difficulty to even recognize it.

Finally, we select 30000 out of 330000 words from vocabulary to uses as choice candidates.

3.1.2 choice produce scheme. For every input word w, we produce the choice by following step:

- `Select Pseudo Answer`: Select one candidate from top 10-50 similar terms with w as pseudo answer.



prisoner (囚犯), asap (煙), peril (岌)
laos (老撾), birthdate (生日), rector (校長)
compendium (概要), testosterone (睾酮), polk (波爾克)
engravers (雕刻機), wacky (古怪), lender (貸款人)
warranty (保證), railroads (鐵路), bata (年輕)

Figure 2: Screen Shot of Front-end

- `Select Other Terms`: Select eight candidate randomly from vocabulary.

3.2 Front-end User Interface

In order to show our works to web users clearly and efficiently, we decide to use **Flask** as our web framework which uses **Jinja2** as the template engine.

At the time user request the server, the server will render the home page and shows it on the browser, which contains `Home`, `Register` and `Login` buttons. All buttons correspond to different pages and services. Let's start with the `Register` page. Firstly, it requires the user to enter his username. After it, he needs to enter a word and post, the server will check if the word exists in our vocabulary. If so, the registration succeeds. The Login service is a bit more complicated, the first step for the user is the same as the registration service - typing the username. This time, the backend server checks the username's existence, and then the authentication process begins. Since our project has 3 different design of authentication choices, the front-end page will show corresponding choices based on the version. Finally, the server returns the result of whether the user successfully logging in.

3.3 Login Handler

This module uses the embedding system as its back-end, packing all the login logic as an interface for the front-end UI. In the perspective of the front-end UI, it simply calls the login handler with `username` , `password` ,and `session id` . This module will then return a boolean value indicating whether the login is denied or not; also, it will return a multiple choice question whose answer has some connection with the `password` while the login process is not yet denied. The front-end UI will present this question to the user and then tell this module which choice the user did choose. Finally, the login handler will calculate the score of this choice with the help of our embedding systems, and again, return a boolean value to notify front-end UI about the status of the login process. When the user passes a fixed amount of questions in a row without being denied, we are convinced that the user indeed has the knowledge of the password and therefore log the user in successfully.

The scoring method is based on the embedding system. The embedding system is able to provide the similarity score between two given words. Thus, the login handler can calculate the similarity score between any choice in a question and the `password`. Having scores of all choices, we then use min-max normalization to map the scores to a [0,1] interval for each question. In each session, we will ask the user 4 questions, and in each question we record the score of each choice. When the average of those 4 normalized scores is smaller then a given threshold, say 0.75, we deny the login; otherwise we accept it.

The exact schemes of those multiple choice question we used and the corresponding thresholds are described in section 4.

4 LOG-IN SCHEME DESIGN

We have designed 3 different schemes in total. Firstly, we use a single word as each choice in the login questions. However, we found that this scheme tends to fail if some powerful attackers try to impersonate the user, detail described in section 5. Therefore, we add up the number of words to 3 in each choice, expecting confusion occurs on attackers. Furthermore, in version 3, we replace plain words with images. Since images could be viewed as set of words implicitly, this version may be think of as an improvement of the previous one. The followings are specific parameter settings of each version :

- `Version 1`: Each question contains 9 choices that consist of one single word each. The average score among all questions must larger than 0.8 to be accepted by our system.
- `Version 2`: Each question contains 5 choices that consist of a 3-word tuple each. The score of a choice is defined as the highest score between the `password` and those words in the 3-tuple. The average score among all questions must larger than 0.8 to be accepted by our system.
- `Version 3`: Each question contains 9 choices that consist of one image each. The images are generated by words in

the embedding system through Google search. Each word is linked to one image, and thus the score of one image is defined as the score of that word behind it. The average score among all questions must larger than 0.6 to be accepted by our system.

5 EXPERIMENT RESULTS

In this section, we conduct real world experiments on all of the three versions. We consider two types of attackers to be our threat models; one type is for random-guessing attackers, and the other is for attackers with embedding systems. Note that in a real world scheme, even if the attacker is so powerful that he/she has an embedding model, it is not necessary to be exactly the same as ours. However, in worst case, a public embedding system may be used by both us and the attackers. It gives great advantage to the attackers to crack our system. Hence, it is still worth studying on the case that the attackers have the same embedding system as ours.

For the case of random-guessing attacker, we set up a bunch of new accounts with different passwords. Then we simulate the attacker by randomly select from all choices in every question uniformly. For the case of attacker with embedding systems, we assume that our algorithm is known by the attacker clearly, but not the `password` only. The attacker will follow the scoring method in our algorithm but using historical logs of choices in one successful login of the real user insdead of his/her `password`. For each question, the attacker will select the highest average score option as the final choice. The next are results of our experiments.

Version 1

attacker type	of success	of tries	success rate
random-guessing	0	10	0%
with embedding	56	60	93.3%

The success rate of the random-guessing attacker is pretty low. However, the attacker with embedding system is powerful. It seems that the leaked information by shoulder-surfing may be still pretty much if the attacker can memorize all the historical logs and get a powerful enough embedding system. Besides, Normal users failed to login 33 times out of total 93 times, the usablity may be low though.

Version 2

attacker type	of success	of tries	success rate
random-guessing	0	10	0%
with embedding	17	17	100%

The design of this version is proposed to confuse the attacker by many unrelated words. For random-guessing attacker, the success rate is still low. However, it seems failed to simply confuse the attacker by redundancy. One possible reason is because the attacker select the word in a 3-tuple with highest score, which may be a good way to filter out the real choice in the tuple. Note that Normal users failed to login 6 times out of total 23 times, and we thought the reason is that there were fewer options for users, the usablity is better than version 1 but still low though.

Version 3

attacker type	of success	of tries	success rate
random-guessing	1	10	10%

In this version, the success rate of the random-guessing attacker slightly increased. It is possibly because of the lower threshold 0.6 used in this version instead of 0.8 in the other version. The word embedding system cannot be applied directly in this version due to the use of images as choices. The attacker may still convert images to words and then use embedding system then; however, we assume that it will add up more redundancy in each choice and thus give the attacker a hard time mimicking the real user.

6 RELATED WORKS

Log-in method based on the use of memory is not new. In [3], multiple cue-response pairs are registered by a user. A user is first asked to come up with a cue and then provide an association as a response. When logging in, a user is provided with a cue and is asked to enter the corresponded response. In [4], a user first register an account with several of his own images. When logging in, a user is challenged with 9 images containing none or only one image that he registered with for a total 4 times. A user must choose the image he registered with if there is one in it. However, the above papers does not take into consideration a fundamental problem we hope to solve, which is to defend against a shoulder surfer. We also take advantage of human knowledge, in our case, language ability which is a skill everyone possesses to make our system even less memory-intensive and to increase user usability.

In [2], the author proposed a log-in method which replace the nodes in pattern lock with images. Every time a user tries to log in, the position of images would change and a user is asked to draw a correct pattern based on his secret which is a sequence of images. While this method can defend against some spyware recording the sequence a user input, it cannot defend against a person peeping directly at the screen.

Several methods have been proposed to solve the eavesdropping problem. In [5], a user is provided with a subset of images from a bigger image set and is asked to memorize images in that subset. When authenticated, a grid of images is provided to the user. The user is asked to move up or left based on whether a image is present in the subset of images he sees when registering. While [5] can effectively defend against shoulder-surfing attack, there is a trade-off between security and human memory capacity. When we require higher security, the amount of information we need to memorize increase. In [1], the proposed method is similar to [2], but it adds more images to the grid provided to a user trying to log in. The consequence is that when drawing a pattern, it passes through other decoy images to confuse a shoulder surfer. A problem to this approach is that the password image is always present on the grid if there is only one set of password. An attacker is able to quickly rule out some images that is not in the password. In our method, we can adjust our threshold to make a attacker hard to gather useful information while maintaining good success rate.

7 CONCLUSIONS AND FUTURE WORKS

Nowadays, personal data protection in smart devices have became more and more important. Despite the fact that our results of this project have proved ineffective, we still learn a lot from it.

The biggest problem of our implementation is that there are too much proportion of unique passwords which fail to log in. Our original object is to design a system to allow users log in successfully in a relative dangerous environment, but user availability should always be the first concern. There are still plenty of challenges for us to overcome if we want to deploy our service online. For example, we could improve our system by limiting the vector space of the word embedding system. It may enhances the similarity between the correct choice and the incorrect ones. Tuning parameters like login threshold is also a critical task, we consider it would improve the usability of the system. In

REFERENCES

[1] Xiuling Chang Xiyang Liu Uwe Aickelin Haichang Gao, Zhongjie Ren. 2010. A New Graphical Password Scheme Resistant to Shoulder-Surfing. *International Conference on Cyberworlds* (Dec. 2010).

[2] Sultan U.J Muhammad K.K Mohsen G Kamran A.A Mudassar A.K, Ikram U.D. 2019. g-RAT | A Novel Graphical Randomized Authentication Technique for Consumer Smart Devices. *IEEE Transactions on Consumer Electronics* 65, 2 (May 2019), 215–223.

[3] Sidney L. Smith. 1987. Authenticating users by word association. *Computersand Security 6* (1987), 464–470. https://doi.org/10.1016/0167-4048(87)90027-7

[4] Koike H. Takada T. 2003. Awase-E: Image-Based Authentication for Mobile Phones Using User's Favorite Images. *Computersand Security 6* 2795 (2003). https://doi.org/10.1016/0167-4048(87)90027-7

[5] D. Weinshall. 2006. Cognitive authentication schemes safe against spyware. *2006 IEEE Symposium on Security and Privacy (SP'06)* (2006).