

Alerting System using Honeyword

Rajkumar E¹, Giridharan S², Senthilkumar M³, Muthaiah V⁴, Pranav S⁵ and Praveen kumar A⁶

¹⁻⁶SRM Valliammai Engineering College, Department of Cyber Security, Kattankulathur, Chennai.

Email: raju.becs39@gmail.com, mailtoogiri@gmail.com, msen1982@gmail.com, muthaiah55555@gmail.com, pranavawm@gmail.com, pk2851764@gmail.com

Abstract— Honeywords are a relatively new approach to enhancing password security by creating decoy passwords that are designed to confuse and deter attackers. The idea is to create multiple decoy passwords, or honeywords, for each real password, and store them in a secure database alongside the actual passwords. To enhance password security, honeywords can be used to create fake passwords to confuse attackers. Multiple fake passwords for each real password are stored in a secure database. When a user logs in, the system checks the entered password to determine if it's a fake one or the real password. If a fake password is used, the system can trigger an alarm or take other steps to stop the attacker. It's important to carefully consider the number and standards for generating honeywords, and to ensure that the system doesn't interfere with legitimate user activity. Honeywords can add another layer of protection, but they should be used along with other security measures.

Index Terms— Passwords, Honeywords, Legacy UI, Honey encryption, Honey-checker, Salting, Sweetword.

I. INTRODUCTION

In the development of web applications, having a login portal is a common feature that allows users to authenticate themselves. Django is a web development system that enables the easy creation of secure and reliable login screens. Its authentication process involves verifying user credentials against the user database and granting access. Constructing a login page with Django involves creating a URL pattern and a corresponding view function that administers the login prompt and processes user data. Django's authentication system also provides additional security features, such as password hashing, reset, and session management, and can be customized to fit app requirements. With Django's built-in authentication mechanism, developers can confidently focus on creating the core features of their program without worrying about authentication security and effectiveness. In order to produce honeywords, a randomization technique is often used to produce passwords that are similar to genuine passwords but distinct enough to be used in place of real passwords. In order to prevent attackers from quickly guessing the genuine password, it is important to generate a sufficient number of honeywords for each real password. The honeywords should be securely kept in a database together with the true passwords once they have been formed. The database should be secured by access controls and other security measures, and the honeywords and real passwords should be hashed and salted to prevent unauthorised access.

The system checks whether the entered password is a honey word or the actual password when a user tries to log in with their password. The system can send an alert, record the event, or take other steps to stop the attacker when a honeyword is used. In general, a key component of integrating honeywords into a web application is creating

honeywords and storing them in a secure database. The system can detect and stop attacks and improve the overall security of the application by creating a large collection of honeywords and storing them securely.

When a Honeyword breach is discovered, it is important to notify the administrator or security team in order to increase Honeywords effectiveness. This allows the group to take immediate action to prevent future access or damage. The system can notify the administrator of a honeyword violation. Depending on the system settings, the warning can appear as a push notification, email or SMS. Information about the incident, such as the attacker's username or IP address, and any other relevant details should be included in the alert. After that, the admin can take necessary actions such as: B. blocking the user account, changing passwords, or performing additional research. The technology has the ability to limit further damage by taking immediate action on Honeyword breaches.

II LITERATURE SURVEY

[1] This architecture connect to the honeyword system to the Internet of Things framework. The security requirements may jeopardise one of the most crucial usability criterion, typo-safety. This is because regular typing errors made by a legitimate user can be mistaken for malicious activity 38.09%–69.67% of the time. This method uses superwords that may not always operate in the same way as the legacy-UI. The strategy most likely belongs in the modified-user interface group as well and show that the aforementioned approach may not sufficiently safeguard view of the enormous amount of associated username-password combinations that can be found when using this threat model, we present a simple technique that ignores any personally identifiable information (PII), such as a person's name or birthday. Using data sets, the suggested method can find username-password pairs that are at least 5 to 22 times more related than expected. Finally this approach gives honeyword-based authentication protocol for IoT devices with limited resources based on a performance comparison.

[2] For external attackers, HPAKE provides honeyword strategies for password leaks. If the server is compromised, the attacker will acquire a password list that contains sweetwords through offline guessing attempts. The attacker most likely utilises HPAKE and a honeyword to get access to the account because they are unable to tell them apart. The authentication is explicitly handled via the key exchange. Using the actual password, only the instruction encrypted by the real session key will be authorised, which also results in production of real key. Although the idea of HPAKE is logical, designing a grounded and secure HPAKE is really challenging. The first is that handling passwords differs between the aPAKE method and the honeyword method. The password cannot leave the client and must be submitted in plaintext to the server for verification. The authentication server specifically runs aPAKE instances, each of which use a distinct password. A session key for an aPAKE with explicit authentication will only be produced when the client password and server password are same.

[3] A fresh attack strategy called a "Multiple System Intersection assault considering Input" is what we suggest. Insufficient security exists for the "Paired Distance Protocol" against this attack scenario. This methodology produces honeywords that are interchangeable with genuine passwords and is effective and useful. Levenshtein distance is used to make sure that a minimum safe distance is maintained between the password and honeywords in order to prevent false detection when a legitimate user types a honeyword due to a typing error. It makes advantage of password databases that hackers disclosed to the public. We demonstrate the exact flatness of our suggested honeyword generating strategies through trials. We contrast our suggested methods with the current honeyword creation methods on a number of metrics to support the claim.

[4] In order to safeguard and manage the user's password, one solution is a password manager. The data is encrypted by the password manager to preserve its security, but they are still susceptible to cyberattacks like brute-force attacks. The user's passwords would all be accessible to the attacker if they successfully guess the master password. As a way to enhance security methods and give password managers protection, honey encryption was one solution that was used. By identifying, stopping, and deceiving data breaches committed by brute-force attackers, the honey encryption method helps password managers solve various security challenges. In order to do detection, the suggested system generates and saves a number of automatically created honey words that are similar to the actual user passwords and which will be contrasted with the entered password. The system recognises the invader if the password matches at least one of the honey words. The defence keeps the intruder from accessing the account and leads him to a phoney account with fraudulent data.

III PROPOSED WORK

A. LOGIN SYSTEM CREATION

There are several steps required to create a login system using Python Django. A new Django project must first be created using the `django-admin startproject` command. Then create a new Django application using `Manage.py startapp` in Python. After building the application, create a user template. Django has a built-in user model that can be extended with additional fields if needed. After defining the user model, you can develop views for the login, authentication, and logout forms.

Django has built-in views and authentication forms that can be used to create a login form and user authentication. You can use Django's `LoginView` and `LogoutView` classes to manage the login and logout processes.

If the inputs match a user in the database, Django's `authenticate` function, which accepts a username and password, produces a user object. User authentication is handled by this function. If authentication is successful, you can utilise the `login` function to allow the user to log in and direct them to the appropriate page.

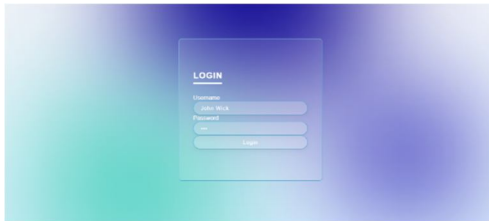


Figure 1. Login Window

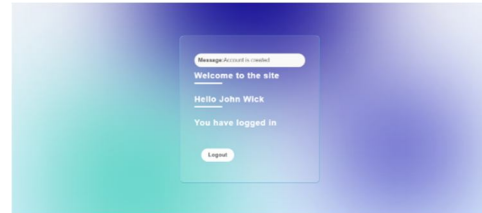


Figure 2. Login Successful

It is necessary to establish the user model, develop views for the authentication and login forms, and use Django's built-in authentication views and forms to manage the login and logout processes.

B. HONEYWORDS GENERATION

To implement honey control in your database, you need to create a mechanism to detect and prevent unwanted access to your private data. To detect suspicious activity, a method known as "honey checking" creates fake content that appears legitimate but is actually designed to trigger an alert upon login. To install Honey Control in your database, you can create dummy tables, rows or columns to attract and trap the attackers. This dummy data may look like real data, but the application does not use it. For example, you can create a fake user account with administrator privileges without actually using the application. If an attacker tries to access this fake account, an unusual activity notification will be sent. Another way to add honey control to the database is to create a separate database that only administrators can access and that contains sensitive data. Any unauthorized attempt to access this database will trigger an alert as only authorized personnel should have access to it. Honey Checker integrates with a database to detect and prevent data breaches by identifying suspicious behavior and alerting administrators. Regularly checking and updating Honey Checker is essential to ensure its effectiveness against new and evolving threats.

```
def generate_similar_honeywords(password, num_honeywords=5):
    honeywords = []
    honeywords.append(password) # add original password as a honeyword
    # generate honeywords by randomly replacing some characters in the password
    for i in range(num_honeywords):
        honeyword = password
        for j in range(random.randint(1, len(password)//2)):
            idx = random.randint(0, len(password)-1)
            honeyword = honeyword[:idx] + random.choice(string.ascii_lowercase + string.digits) + honeyword[idx+1:]
        honeywords.append(honeyword)
    return honeywords
```

Figure 3. Code to generate the honeywords

```
from django.db import models

# Create your models here.

class Honeyword(models.Model):
    password = models.CharField(max_length=255)
    honeyword = models.CharField(max_length=255)
```

Figure 4. Creation of the honeywords model

C. ALERTING ADMIN

The list of honeywords can then be created for each user account. These honeywords should be unique to each user and challenging to figure out. When a honeyword is entered, a notification ought to be sent to the user and the system administrator. The alert system should be set up to immediately notify the user that their account may have been compromised if a honeyword is typed. The system administrator should also be informed so they may look into the breach and fix it. A honeywords alert system can aid in stopping data breaches by spotting and alerting users and administrators to unusual behaviour. To ensure that the honeywords remain effective against evolving threats, they should be examined and updated frequently. Users should be reminded to use strong passwords and to change them frequently in order to further aid in preventing unauthorised access to their accounts.

D. CODE USED TO SEND EMAIL TO THE ADMIN

```
“if Honeyword.objects.filter(password=pass1).exists() and password1!=pass1:
    messages.warning
(request, 'WARNING: The password you entered is a honeyword. Please enter your real password.')
message = 'WARNING: The password you entered is a honeyword. Please enter your real password.'
subject = 'Honeyword Detected'
send_mail
(
    subject,
    message,
    settings.DEFAULT_FROM_EMAIL,
    [settings.ADMIN_EMAIL],
    fail_silently=False,
)”
```

IV. CONCLUSION

Building secure online apps requires setting up a login page using Django. Django has built-in support for user authentication and a robust authentication system with tools for managing sessions and password hashing. These features enable developers to design safe and streamlined login screens that safeguard user data and credentials. A database's use of honeywords is a reliable method for spotting and stopping attackers who are attempting to compromise a system. The system can identify attempts to guess passwords or breach the system by creating a set of honeywords for each real password and securely storing them in a database. A web application's overall security can be improved with honeywords, which also add an extra layer of defence against unauthorised access.

REFERENCES

- [1] N. Chakraborty, M. Mukherjee, J. Li, M. Shojafar and Y. Pan, "Cryptanalysis of a Honeyword System in the IoT Platform," in IEEE Internet of Things Journal, vol. 9, no. 4, pp. 2614-2626, 15 Feb.15, 2022, doi: 10.1109/JIOT.2021.3080676
- [2] W. Li, P. Wang and K. Liang, "HPAKE: Honey Password-Authenticated Key Exchange for Fast and Safer Online Authentication," in IEEE Transactions on Information Forensics and Security, vol. 18, pp. 1596-1609, 2023, doi: 10.1109/TIFS.2022.3214729.
- [3] Akshima, D. Chang, A. Goel, S. Mishra and S. K. Sanadhya, "Generation of Secure and Reliable Honeywords, Preventing False Detection," in IEEE Transactions on Dependable and Secure Computing, vol. 16, no. 5, pp. 757-769, 1 Sept.-Oct. 2019, doi: 10.1109/TDSC.2018.2824323
- [4] A. AlMuhanna, A. AlFaadhel and A. Ara, "Enhanced System for Securing Password Manager Using Honey Encryption," 2022 Fifth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU), Riyadh, Saudi Arabia, 2022, pp. 150-154, doi: 10.1109/WiDS-PSU54548.2022.00042.
- [5] Hire and J. Katti, "Security of User Credentials on Web Portals," 2021 International Conference on Computing, Communication and Green Engineering (CCGE), Pune, India, 2021, pp. 1-4,doi: 10.1109/CCGE50943.2021.9776476
- [6] D. Wang, Y. Zou, Q. Dong, Y. Song and X. Huang, "How to Attack and Generate Honeywords," 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 966-983, doi: 10.1109/SP46214.2022.9833598
- [7] J. Jeong, D. Woo and Y. Cha, "Enhancement of Website Password Security by Using Access Log-based Salt," 2019 International Conference on Systems of Collaboration Big Data, Internet of Things & Security (SysCoBioTS), Casablanca, Morocco, 2019, pp. 1-3, doi: 10.1109/SysCoBioTS48768.2019.9028012.
- [8] V. Papaspirou et al., "Security Revisited: Honeytokens meet Google Authenticator," 2022 7th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Ioannina, Greece, 2022, pp. 1-8, doi: 10.1109/SEEDA-CECNSM57760.2022.9932907