1. The HoneyWords [2] aims to enhance protection of password-based authentication systems. Design and implement (in any language) a simple version of HoneyWords [1]. Your implementation should have at least the following features:

- **Create new accounts**
- **Authenticate to the front-end server by sending their (username, password) pairs.**
- **Raise an alarm on detection of malicious logins (i.e., when there is evidence that an incorrect, cracked, password is used).**

Communication between entities in your system must be done over TCP. Demonstrate that the system works, and analyse its security. (Keep in mind that an adversary knows how your system works internally.)

https://people.csail.mit.edu/rivest/pubs/JR13.pdf

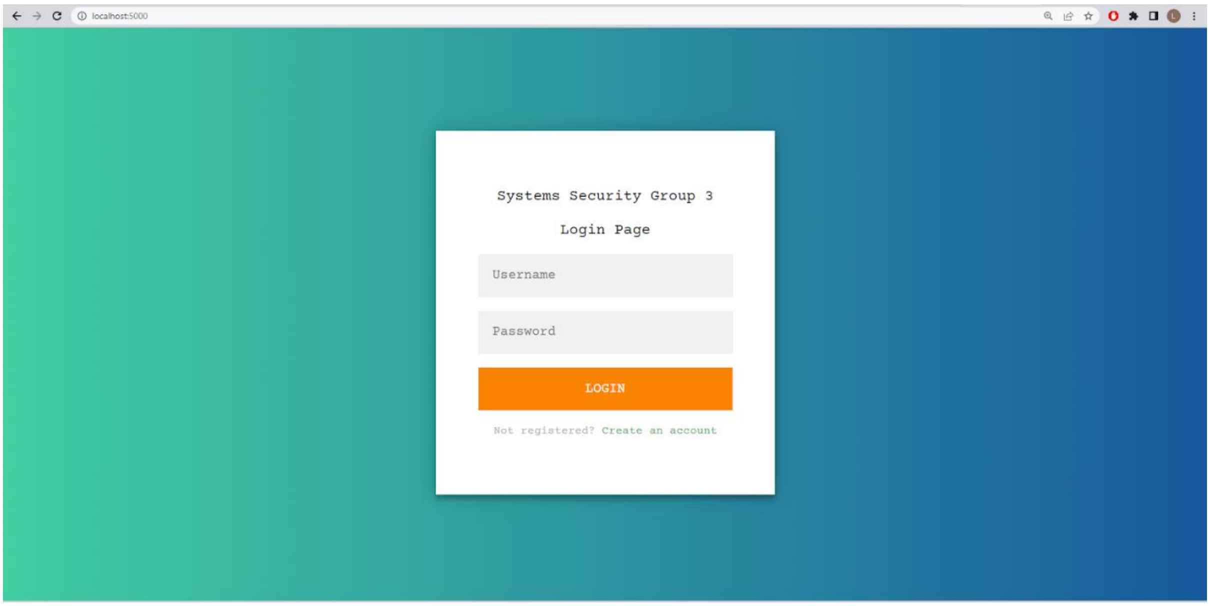
HoneyWords attempts to detect malicious logins by adding several more honeyword hashes in addition to the hash of the real password. If an attacker were to steal the database with the hashes and usernames, they may not know which is the real hash. If they were to login with the honeyword hash, their attempt will raise an alert.

For our submission, HoneyWords was implemented using a simple web application which allows users to create new accounts and authenticate to it using their username and passwords. In this project, our method for generating HoneyWords follows the “chaffing with a password model” method presented in the paper, where honeywords are generated using a probabilistic model of real passwords. Thus, if an attacker tries to log in using a common password (such as in a dictionary attack), it will tally with a honeyword and prompt an alert.

The repository for this project can be found on Github, via the following link:
https://github.com/kertzlim96/SS-HW4-Q1

Local Web Server

After running the webserver.py script, the following web application can be visited via http://localhost:5000. This brings the user to the default Login Page:

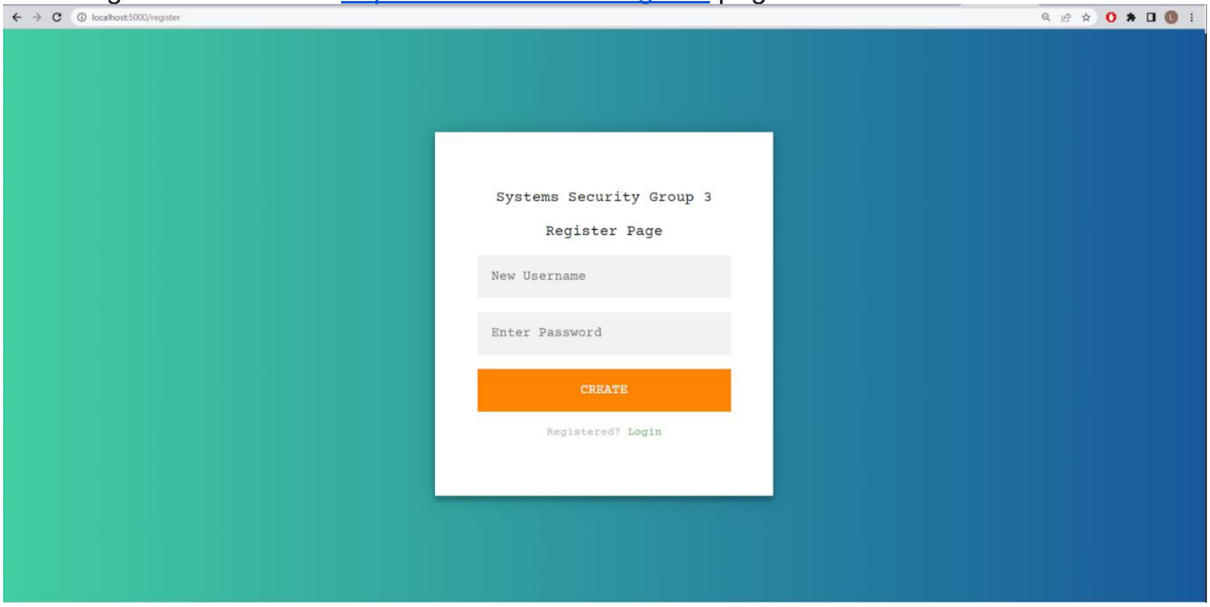


Systems Security Group 3

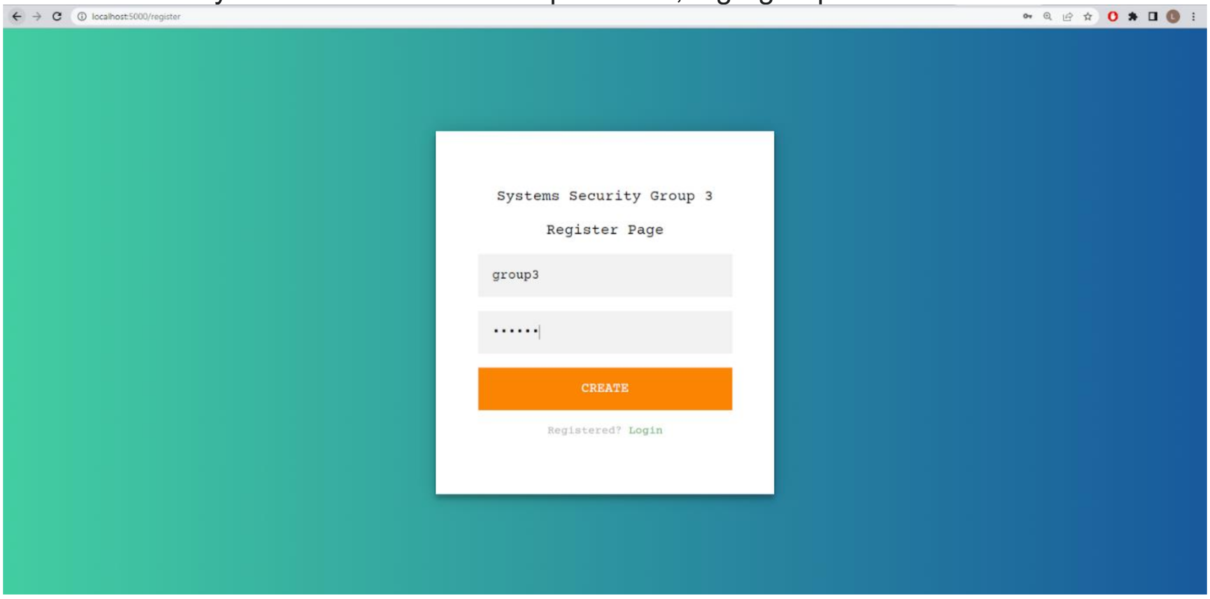
Login Page

Create new accounts

If the user does not have an account registered, the user can click on “Create an account” and this brings the user to the http://localhost:5000/register page:



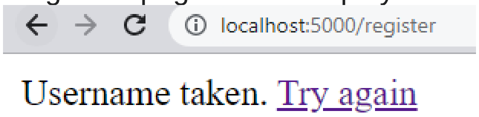
The user can key in a new username and password, e.g. “group3” shown below:



group3

••••••

When registering a new user, the username will be checked to see if it already exists. If the user already exists, the following error page will be displayed:

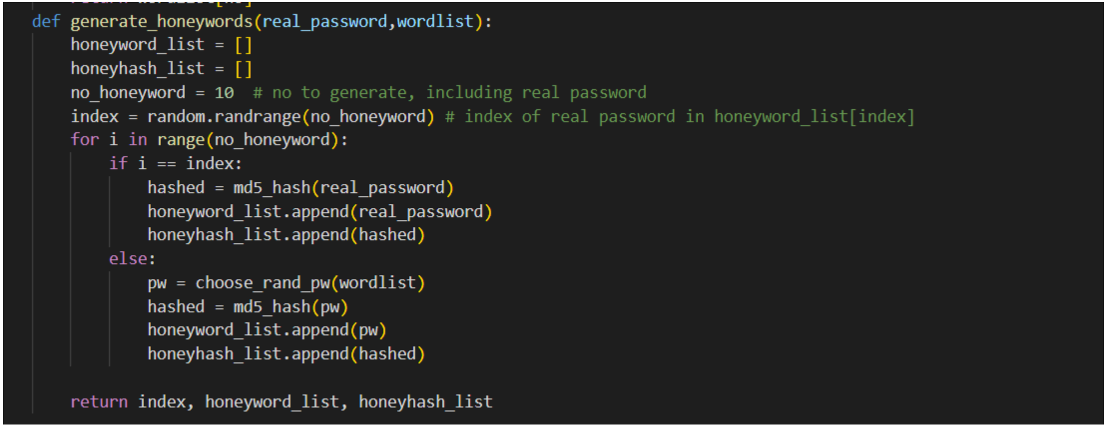


Username taken. Try again

If the username doesn’t exist, *gen_and_add_new_user_db()* function is used to create a new user. Honeywords will be generated with the *generate_honeywords()* function. The username, hashes, and index for the real hash are then stored to a csv file.

Generating honeywords

generate_honeywords() generates 9 honeywords to go along with the password, hashes it and puts it in a random order in the list. The honeywords are randomly selected from a wordlist, in this case, the 10k-most-common wordlist.

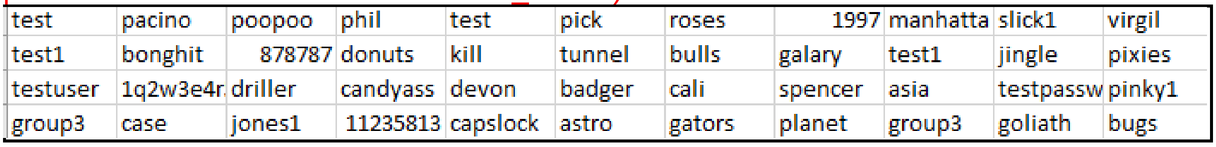


```
def generate_honeywords(real_password,wordlist):
    honeyword_list = []
    honeyhash_list = []
    no_honeyword = 10  # no to generate, including real password
    index = random.randrange(no_honeyword) # index of real password in honeyword_list[index]
    for i in range(no_honeyword):
        if i == index:
            hashed = md5_hash(real_password)
            honeyword_list.append(real_password)
            honeyhash_list.append(hashed)
        else:
            pw = choose_rand_pw(wordlist)
            hashed = md5_hash(pw)
            honeyword_list.append(pw)
            honeyhash_list.append(hashed)

    return index, honeyword_list, honeyhash_list
```

When the “group3” user is registered, the following databases are updated:

a) pass_db.csv

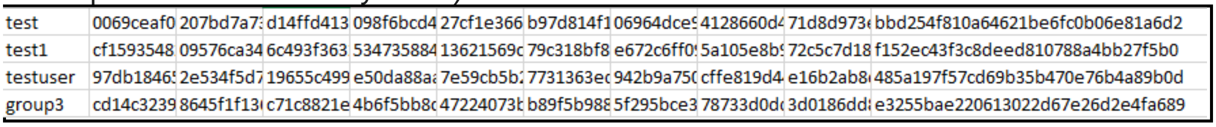
In pass_db.csv, the username (first column) is stored along with the password and honeywords (column 2 to 10) where “group3” is our actual password and the rest are honeywords. (This password plaintext is only used for testing and explanation purposes, the password hash is used instead in Hash_db.csv)



test	pacino	poopoo	phil	test	pick	roses	1997	manhatta	slick1	virgil
test1	bonghit	878787	donuts	kill	tunnel	bulls	galary	test1	jingle	pixies
testuser	1q2w3e4r	driller	candyass	devon	badger	cali	spencer	asia	testpassw	pinky1
group3	case	jones1	11235813	capslock	astro	gators	planet	group3	goliath	bugs

b) hash_db.csv

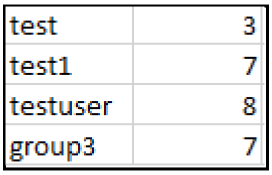
This database contains all the hashes in the same index order as the pass_db.csv (includes actual passwords and honeywords)



test	0069ceaf0	207bd7a7	d14ffd413	098f6bcd4	27cf1e366	b97d814f1	06964dce	4128660d	71d8d973	bbd254f810a64621be6fc0b06e81a6d2
test1	cf1593548	09576ca34	6c493f363	534735884	13621569	79c318bf8	e672c6ff0	5a105e8b	72c5c7d18	f152ec43f3c8deed810788a4bb27f5b0
testuser	97db1846	2e534f5d7	19655c499	e50da88a	7e59cb5b	7731363e	942b9a75	cffe819d4	e16b2ab8	485a197f57cd69b35b470e76b4a89b0d
group3	cd14c3239	8645f1f13	c71c8821e	4b6f5bb8	47224073	b89f5b98	5f295bce3	78733d0d	3d0186dd	e3255bae220613022d67e26d2e4fa689

c) index_db.csv

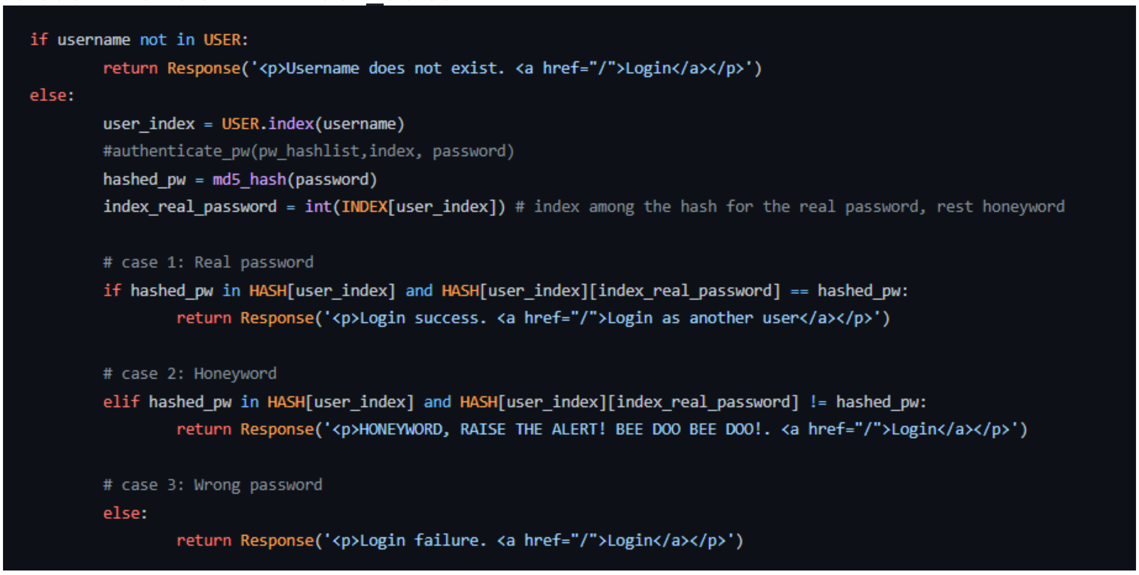
This database shows the column/index in which the actual password is stored. The hashes and index are stored in different places so in the event that the hashes are stolen, the attacker will not know which is the real hash.



test	3
test1	7
testuser	8
group3	7

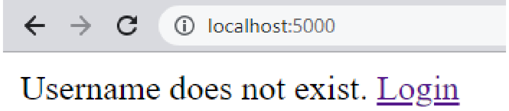
In an actual implementation of HoneyWords, there will be a separate secure server, also known as a “honeychecker”, which will store these hashes and indexes and the web server will communicate with the honeychecker via encrypted channels to perform the authentication.

Authenticate to the front-end server by sending their (username, password) pairs:
After reading from the index_db.csv and hash_db.csv, the entered username will be checked to see if it exists within hash_db.csv:



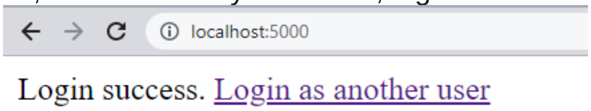
```
if username not in USER:
        return Response('<p>Username does not exist. <a href="/">Login</a></p>')
else:
        user_index = USER.index(username)
        #authenticate_pw(pw_hashlist,index, password)
        hashed_pw = md5_hash(password)
        index_real_password = int(INDEX[user_index]) # index among the hash for the real password, rest honeyword

        # case 1: Real password
        if hashed_pw in HASH[user_index] and HASH[user_index][index_real_password] == hashed_pw:
                return Response('<p>Login success. <a href="/">Login as another user</a></p>')

        # case 2: Honeyword
        elif hashed_pw in HASH[user_index] and HASH[user_index][index_real_password] != hashed_pw:
                return Response('<p>HONEYWORD, RAISE THE ALERT! BEE DOO BEE DOO!. <a href="/">Login</a></p>')

        # case 3: Wrong password
        else:
                return Response('<p>Login failure. <a href="/">Login</a></p>')
```

If it does not exist, a “Username does not exist” error will be displayed:



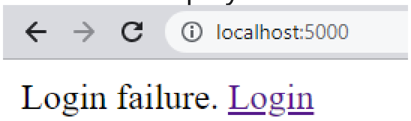
Username does not exist. Login

If it does, the entered password is hashed and compared with the stored hashes. If the hash exists and is the real hash, as indicated by index.csv, login successful:



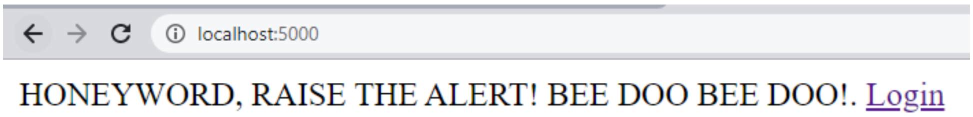
Login success. Login as another user

If the hash of the entered password does not match with the actual password or any of the honeyword hashes, a login failure error is displayed:



Login failure. Login

Otherwise, if the hash exists but it is not the hash of the actual password, an alert will be generated:



HONEYWORD, RAISE THE ALERT! BEE DOO BEE DOO!. Login

In the event where the account credentials were exfiltrated or in a dictionary attack, by logging in with the username and honeyword, it will activate the honeyword page which will alert the sysadmin that the password database has been exfiltrated.