Secure System Design (CS392)

Name : Vaibhav Gakhreja **Assignment number : 2**
Roll Number : 1801cs58 **Date of submission : 11 April 2021**

Task 1 : Implement Erguler's honeyword based scheme in a linux environment

Imran Erguler's paper addresses majorly two issues of the original honeyword paper by Juels and Rivest :

1)The honeyword generator function must be flat , meaning that honeywords should not be distinguishable from sugarword, so that it is difficult for the adversary to find the correct password.

2)The original honeyword storage scheme consumes very large memory just to store duplicate passwords.

So this paper proposes a scheme which increases the flatness of the honeyword generator, as well as it reduces the storage cost.

Original scheme by Juels and Rivest:(legacy UI implementation)
If a new user is signing up, then he enters his username and password just like normal signup portals. And internally the server uses a predefined scheme to generate multiple duplicate passwords which resemble the original password. It stores all(original + generated) hashed passwords and stores the index of the correct password of the user in a separate server.

New scheme by Imran Erguler:
1)To improve flatness:
Using algorithmic strategy to generate new passwords is bad because users do not choose passwords randomly, so the generated password becomes easily distinguishable from the original password. So we use passwords of other users as honeywords for a new user. The honeyword are chosen randomly from the set of old passwords and the index of sugarword among sweetwords is also chosen randomly.

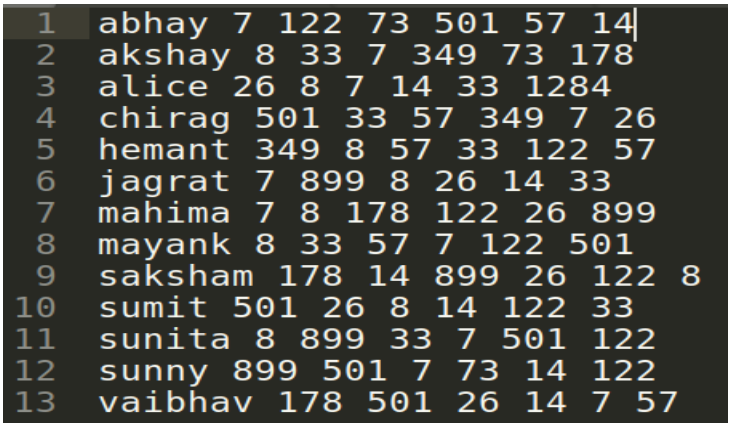
2)To reduce storage requirement:
Instead of storing the hashed passwords with username in a single table, we assign unique numbers to each of the passwords and store a set of those numbers instead of a set of strings for each username. And we maintain another table which stores the assigned numbers along with the hashed password to which it was assigned. Numbers take less space to store than string, hence repeated numbers consume much less space than repeated strings.

Observation : it also solves the problem of inconsistency by storing the actual information at one place and referring to it by its id. Suppose if a user decides to change its password then with the original scheme it would have been very time consuming and error prone to update

the password at each place it was used. But with the new scheme it should be updated only at one place. **Hence the new database design is normalised.**

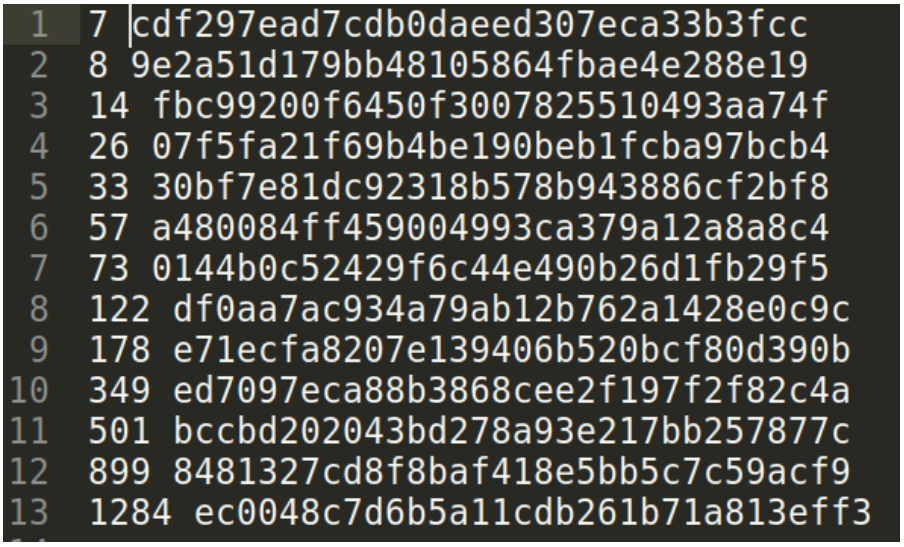
Implementation details:
File structure :
1)F1.txt: stores username with all its sweetword indices.

Screenshot :

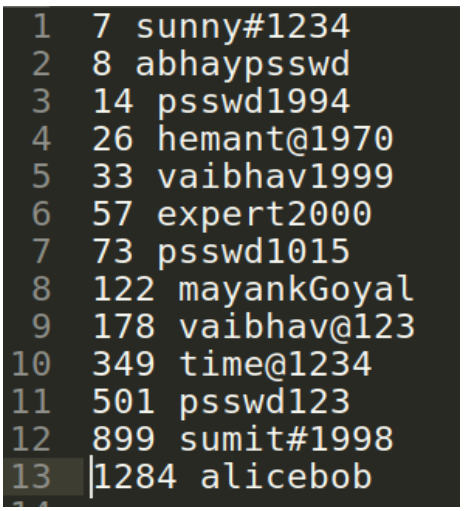


2)F2.txt: stores hashed passwords with the unique random numbers (index) assigned to them.

Screenshot:



3)F2_unhashed.txt: stores unhashed passwords with the unique random numbers (index) assigned to it.(as I have assumed that the attacker has cracked the hashes so he can access the raw text password).
Screenshot:

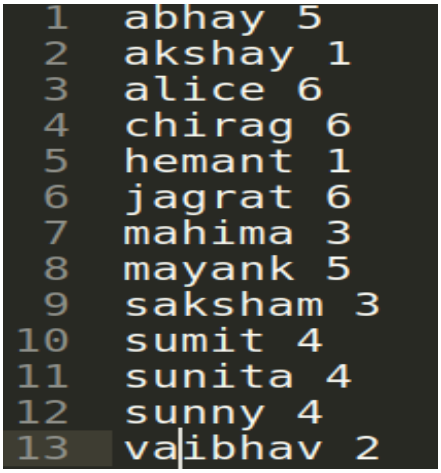


4)temp.txt: it is used by md5 hashing calculator as a temporary storage.

5)hashFile.txt: it is used by md5 hashing calculator to write its result.

6)result.txt: honeychecker server uses it to store its result and it is read by the honeyword generator program thereafter.

7)honeyserverData.txt: stores the indices of sugarwords among all the honeywords for all usernames.
Screenshot:



8)honeywordGenerator.c: it interacts with users to create new users or to servelogin requests.

9)honeywordServer.c: it interacts with the honeywords Generator program to check whether the login request hit a honeyword or sugarword.

Assumption : I have assumed that the attacker has already cracked the hashes , so I have maintained another table F2_unhashed.txt where I have stored the index of password along with original(text) password.

Important features of this implementation:

1)As described in the paper I have assumed that there are at least k-1 entries in the database before starting to use the program. Hence I have manually entered data for 13 users in the tables before testing.

2)The file honeyserverData.txt is being edited and read only by the honeychecker server(filename = honeywordServer). Because as described in the original paper (Juels and Rivest), the information about the index of correct password among the sweetwords is maintained and retrieved only by a separate server, which only answers one type of GET request i.e. Is the pair <username,index> is the correct combination or not. And only one type of POST request i.e. add the pair <username, sugarword index> to the table honeyserverData.txt.

Hence , HoneywordServer program supports two types of queries :

1) add the pair <username, sugarword index> to the table honeyserverData.txt.
2) check whether the pair <username,predicted sugarword index> is a correct pair or not by referring to the honeyserverData.txt file.

3)The previous passwords which are used as honeywords are selected randomly and the position of the correct password among all the sweet words is also selected randomly.

4)The files F1.txt and F2.txt which stores the username and corresponding set of sweet words are sorted in ascending order of their primary keys. F1.txt is sorted as per the usernames and F2.txt is sorted as per the index assigned to the passwords. This is done to ensure that the adversary does not find any patterns in the database. So after adding a new entry , I have sorted both tables F1 and F2.

5)I have used fork and exec functions to call the honeychecker server from the honeyword generator program. And I have used the system function to execute the shell command for converting passwords into md5 hashes. I have modified the command so that the hashFile gets overwritten each time a new hash is generated. New command : md5deep temp.txt > hashFile.txt. I have written comments in my code to explain each implementation detail comprehensively.

6)My program takes the number k (the number of sweet words maintained for user) as input. So that the number of sweet words is flexible as per user requirement.

Sample execution:

1)Compile and run the source codes.

Compile the honeychecker server : gcc honeywordServer.c -o honeywordServer
Compile the honeyword generator : gcc honeywordGenerator.c -o honeywordGenerator
Run the honeyword generator : ./honeywordGenerator

Note that the honeyword server is not directly used by the user, its executable is called by the honeyword generator program.

Screenshot :

```
vaibhavgakhreja@vaibhavgakhreja-Inspiron-5570:~/Desktop/Academics/Vaibhav Gakhr
eja/security/post midsem/honeyword papers/assign2$ gcc honeywordServer.c -o hon
eywordServer
vaibhavgakhreja@vaibhavgakhreja-Inspiron-5570:~/Desktop/Academics/Vaibhav Gakhr
eja/security/post midsem/honeyword papers/assign2$ gcc honeywordGenerator.c -o
honeywordGenerator
vaibhavgakhreja@vaibhavgakhreja-Inspiron-5570:~/Desktop/Academics/Vaibhav Gakhr
eja/security/post midsem/honeyword papers/assign2$ ./honeywordGenerator
to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
```

2)Adding a new user:

1) Type 0 and press enter.
2) Enter username
3) Enter password
4) Enter the number of sweet words to be stored for this user.

Screenshot:

```
to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
0

enter username
karan

enter password
karan@1994

enter the number of sweetwords(k)
6

sweetword indices for this user:->
8
7
57
1946
26
14

to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
```

3)An attacker trying to login:

1) Type 1 and press enter.
2) Enter username.
3) Guess password from the displayed list of sweet words for this user and enter that password.
4) Result is displayed on the screen.

Screenshot showing execution of a successful login:

```
to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
1

enter username
vaibhav

the sweetwords for this username are :->
vaibhav@123
psswd123
hemant@1970
psswd1994
sunny#1234
expert2000

enter the guessed password
psswd123

this is an authentic user

to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
```

Note that I have assumed that the attacker has information about the sweet words of the user and he has already cracked the hashing, so I have displayed the sweet words in raw text form.

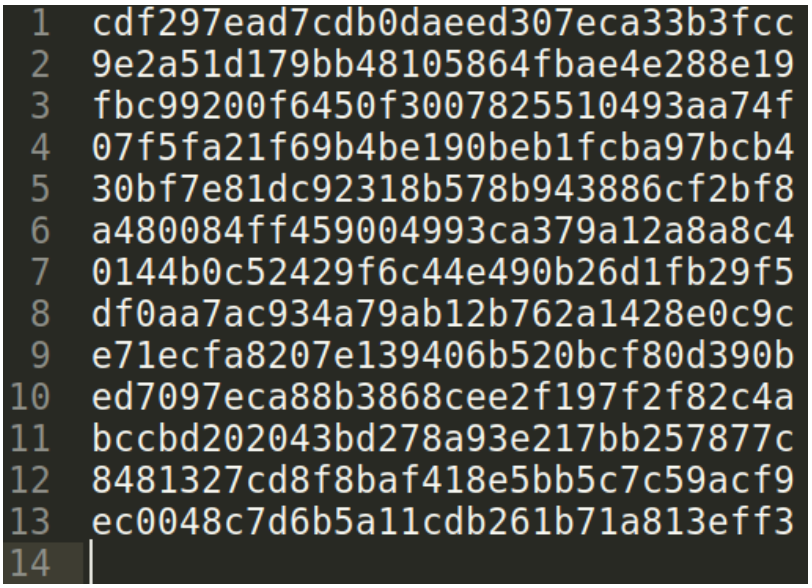
Screenshot showing execution of a failed login:

```
to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
1

enter username
vaibhav

the sweetwords for this username are :->
vaibhav@123
psswd123
hemant@1970
psswd1994
sunny#1234
expert2000

enter the guessed password
vaibhav@123

password file compromise detected : the user is using a honeyword as a passwor
d, so take appropriate action(block this account or block the entire system)

to create a new user : enter 0
to try to login as a existing user : enter 1
to exit : enter -1
```

Task 2 : Use of Password Cracking Tool

Installed the john ripper tool as guided by the pdf.

1)Made a new file passwordFile.txt and wrote 13 md5 passwords saved in my honeychecker server.
Screenshot:



2)Run the command : john --wordlist=rockyou.txt passwordFile.txt . to crack the md5 hashes

3)checked result using the command : sudo john --show passwordFile.txt
None of the hashed passwords could be cracked.
Screenshot: