Computer Security

CS5352

Erick Garcia

Brute Force / Dictionary Attack

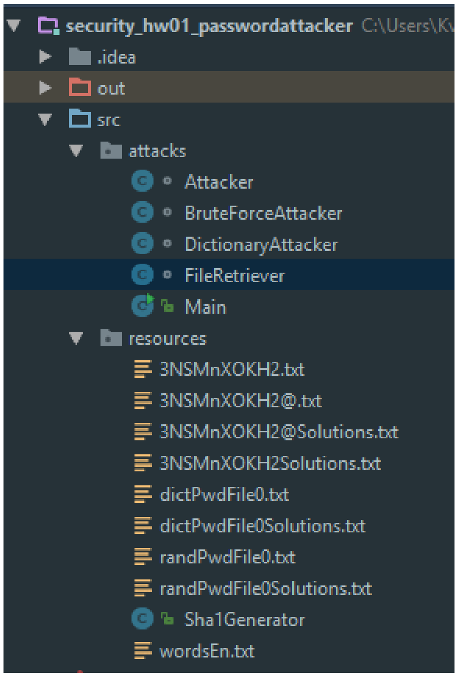
02/12/2017

Erick Garcia 02/12/2017
Dictionary / Brute Force Attack

1. The URL for the word list you used for this assignment. In addition,
provide URLs for two other interesting word lists you have found, with a one or two lines description of the list:

- http://www-01.sil.org/linguistics/wordlists/english/
- http://www.math.sjsu.edu/~foster/dictionary.txt
  - English dictionary with odd combinations of characters, useful probably for cracking passwords with dictionary words + random characters.
- https://github.com/dwyl/english-words
  - A text file containing 355k English words

2. This is a screenshot of my project structure:

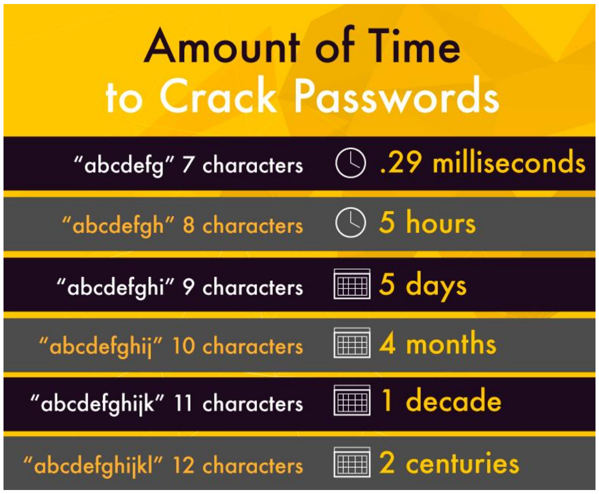


```
▼ security_hw01_passwordattacker C:\Users\K
  ▶ .idea
  ▶ out
  ▼ src
    ▼ attacks
        Attacker
        BruteForceAttacker
        DictionaryAttacker
        FileRetriever
        Main
    ▼ resources
        3NSMnXOKH2.txt
        3NSMnXOKH2@.txt
        3NSMnXOKH2@Solutions.txt
        3NSMnXOKH2Solutions.txt
        dictPwdFile0.txt
        dictPwdFile0Solutions.txt
        randPwdFile0.txt
        randPwdFile0Solutions.txt
        Sha1Generator
        wordsEn.txt
```

- Resources Package:
  The resources package includes all of the files that were used on this project. I also included the Sha1Generator which hashes a String into Sha1.
- Attacks Package
  The main package is the attacks package. Let's start with the FileRetriever class. This class only stores the random passwords and salted passwords and stores them into a 2-dimensional array, and also stores the Dictionary from wordsEn.txt into an ArrayList.
  - Attacker (Superclass): This class stores the information from the password files we have to crack and stores them into a 2-dimensional array, an optional dictionary if the attack is a dictionary attack.
    The Methods consists of validatePassword() which consists of checking if our password guess equals the hashed password from the file we received, and getEstimatedTime() which returns the amount of time it took to crack a password.
  - Dictionary Attacker (Subclass): This class has only one method which is attack() it consists of going through each user row from the file and comparing their hashed password with a word from the dictionary.

- Brute Force Attacker (Subclass): This class contains an array of the possible characters that a password can have, and an array list of the current combinations based on the number of characters. The method attack builds the first possible combinations by adding each of the character as a separate combination in the ArrayList. Afterwards this becomes an iterative process in which we iterate through the ArrayList and to each combination we append each of the possible characters to go through every combination. This process increases the time it takes each time it iterates

3. The problems I encountered were for the most part in the brute force attack. It was probably due to the implementation of my brute force algorithm but I ended up with a GC out of memory error from java, probably because the process ran out of heap space, in the end I was only able to crack 4 passwords. From my perspective and the amount of time we had for the project I think it would have been impossible to crack all of the passwords because of the following: 1. A truly random password is hard to crack, 2. The amount of characters available were 64 so the number of combinations with each iteration increases substantially, I will attach a fun infographic I found on the internet.



Amount of Time to Crack Passwords

"abcdefg" 7 characters	.29 milliseconds
"abcdefgh" 8 characters	5 hours
"abcdefghi" 9 characters	5 days
"abcdefghij" 10 characters	4 months
"abcdefghijk" 11 characters	1 decade
"abcdefghijkl" 12 characters	2 centuries

4. Results:

- Dictionary Attack cracked passwords:
  james24_Bhi, misdoubted
  linda24_Bhi, tryouts
  sofia24_Bhi, macaques
  santiago24_Bhi, vined
  isabella24_Bhi, drachmae
  diego24_Bhi, shammied
  robert24_Bhi, pavane
  mary24_Bhi, canceling
  patricia24_Bhi, anointers
  daniela24_Bhi, chorus

- Brute Force Attack cracked passwords:
  isabella24-Bhi, D
  linda24-Bhi, u1
  mary24-Bhi, JQi

sofia24-Bhi, _zSD

5. The salting technique is a one-way function that hashes a password or passphrase. In this case the salt was appended to the end of the password string before hashing into Sha1. In a secure system hashing is a must because if a malicious user can see the data from the database and he cracked a password of a user. E.g. Password was 12345 if we hash that into sha1 and another user has password 12345 the hash will turn out to be the same in the end, so the hacker will know if two users have the same password. If we per user salting the case would be this: User1: 12345+salt1, User2: 12345+salt2. Even though the passwords are the same the hash in the end will be different, making the system a more secure one.
6. References
    - My Code: https://bitbucket.org/egarcia87/security_hw01_passwordattacker/
    - https://en.wikipedia.org/wiki/Salt_(cryptography)
    - https://www.betterbuys.com/estimating-password-cracking-times/