Practicing with an offline dictionary attack
HW2 - CNS Sapienza

Lombardo Andrea 1893440

14/11/2019

1 Introduction

Nowadays there're different cyber-attacks which strike people using different methodologies for cracking password. Applying strategies like Brute force or dictionary attacks are included in the set cited above. They are very dangerous because the real target is not control another account in case for example in social media but they manage to break the principle of authentication . Indeed in the way they are able to discover personal/secret information about you.

The paramount technical attack on password-based authentication system is the dictionary attack. It refers to the general technique of trying to guess passwords, by running through a list of instance, in general it's used a list of words that we call dictionary. The dictionary attack is defined as offline because the attacker carry out all the needed information and then performs the computations offline. In general it could be applied under two situations:

- it is used to find out the decryption key for obtaining the plaintext from the ciphertext(which is what we're interested in)
- it can be used to guess password

For improving speed computation of the attack there are specialized versions used for cracking password where the dictionary are sorted by frequency or alphabetical words. The probability of the success of this attack depends not only by computational power but also by the used dictionary and the rules applies for it, for example a tool that apply to the tested password. Further, password quality plays a key role in this sense, so it's not only important the length but also the sequence of the character of the password.

2 PBKDF2

The name PBKDF2 is for PASSWORD-BASED KEY DERIVATION FUNCTION and its target is to increase the computational cost for a possible brute force attack. Despite the fact that it's a key derivation function where giving in input a password is applied an hash algorithm in more rounds where the standard suggests at least 1000 instead big company as Apple at least 2000 for achieving its goal(so for protecting against brute force attack and dictionary attack). Due to the fact that these attacks increase with iterations it's verified that there's a practical limit on the iteration count visible in the a decrypted function. A possibility for users that in same case could be a shortcut could be the usage of salt which protect against attackers from precomputing a dictionary of derived keys.

An alternative approach, called key strengthening, extends the key with a random salt.

Nowadays this important technique is used in different systems:

- Wi-Fi Protected Access (WPA and WPA2)
- WinZip's AES Encryption scheme
- Apple's iOS mobile operating system, for protecting user passcodes and passwords.
- Mac OS X Mountain Lion for user passwords
- The Django web framework, as of release 1.4

3 Design

The main goal of this report is to analyze performance and the success of a dictionary attack: in detail giving a encrypted file provided and the hints that the key word is an English word and the decrypted algorithm which should be used is AES with 192 of key size and with the usage of CBC as cipher mode.
I decide first to download on the net a English dictionary text file from the following url:
http://www.gwicks.net/dictionaries.htm

3.1 Dictionary

I decide to first try with the smallest one and **engmix** which contain 84.000 words and the size of the file is 223kb.
After download it I study what I should aspect and what are the benefits to use **-pbkdf2** instead of salt, analyzing different sources on the net.

With OpenSSL and Bash, I can do a quick check of the passwords included in the content in each line of the dictionary text used to protect this file ciphertext.enc. I create a bash file following the content of Linux Magazine behind in the section of **Auditing Encryption Passwords:**

http://www.linux-magazine.com/Online/Features/OpenSSL-with-Bash.
Here I establish how can I do the following job: I try the decryption routine and I store the returned file with the name of own **password-tried.txt** in the same directory of the working space. A second routine is calling according the syntax if-else applied to the bash script verify if the file created is an ASCII text or not. This is the real policy to identify which is the correct password. If it's not right I remove the file. For proving it I use the following line:

if file "${password}.txt" | grep 'ASCII text';

where first I take the file and then I analyze with the grep routine if it contains ASCII text otherwise I remove the file with **rm** call with the filename.

4 Result

About the result we can say that the key word found is **learning** and when it's found there's a cat of the file which let us to analyze the content which correspond to the plaintext of a Shakespare's sonnet in detail the text discovers is Hamlet. About measurement I test that the work cost a lot in fact the right key is found out in **20 minutes and 18 seconds** instead all the file of the dictionary is analyze in more than **38 minutes and 40 seconds** .

5 Conclusion

In conclusion, we can say that could be interesting making different tests with dictionaries with a disparity size, analyzing them and making a comparison for analyzing performance, it could be also interesting using another way of designing the problem starting from the end of the dictionary or from the middle with different encryption problem or pick word from the collection according an algorithm which chooses once random key from the dictionary. Another important issue that could be brought some change is the usage of tdf-idf sorted of the dictionary .

However we should say that for performance reason it's better to use or adding the salt to the password due to the fact it reduce the success of the attack because multiple password should be tested individually.

References

[1] https://unix.stackexchange.com/questions/507131/openssl-1-1-1b-warning-using-iter

[2] http://www.linux-magazine.com/Online/Features/OpenSSL-with-Bash

[3] https://www.xypro.com/data-protection/how-to-resist-a-dictionary-attack/

[4] https://www.eyeonpass.com/yes-you-should-be-worried-about-dictionary-attacks/

[5] https://en.wikipedia.org/wiki/PBKDF

[6] https://en.wikipedia.org/wiki/List_of_PBKDF2_implementations

[7] https://it.wikipedia.org/wiki/Password_cracking

[8] https://it.wikipedia.org/wiki/Attacco_a_dizionario

[9] https://hackerstribe.com/vocabolario/attacco-dizionario-e-forza-bruta/

[10] http://www.wikibit.it/a/cosa-significa-attacco-a-dizionario-1809/

[11] https://www.openssl.org/docs/man1.1.1/man1/enc.html

[12] http://www.linux-magazine.com/Online/Features/OpenSSL-with-Bash

[13] https://www.openssl.org/

[14] https://www.tldp.org/LDP/abs/html/special-chars.html