# Practicing with an offline dictionary attack

HW2-CNS Sapienza

Manuel Ivagnes 1698903

14/11/19

## 1 Introduction

Even knowing the algorithm used for the encryption, it is computationally hard to decrypt a given ciphertext generated with a good cipher and a long key. The purpose of the homework is to decrypt a given ciphertext using an offline dictionary attack, which is a form of brute force attack that use hundreds or sometimes millions of likely possibilities, such as words in a dictionary, to determine the decryption key or passphrase used to encrypt the ciphertext.

The standard brute force attack, systematically generate new keys, trying all the possible combinations, giving an huge amount of possible values. Using the standard ASCII for the password there are 95 printable characters, which mean $95^k$ iterations to consider all the possible entries, where $k$ is the length of the password.

The dictionary attack, instead, tries only those possibilities which are deemed most likely to succeed. Indeed, many people have a tendency to choose short passwords that are ordinary words or common passwords, or simple variants obtained, for example, by appending a digit or punctuation character.

In this case, for instance, I used an english dictionary with 10000 entries, which require a number of iterations almost equal to consider just 2 letters with the previous method. (the link to the dictionary is in the bibliography).

## 2 Model

The message has been encrypted with OpenSSL 1.1 using:

- Cipher: AES
- Key length: 192 bits
- Mode: CBC
- Other parameters: -pbkdf2

It is also known:

- The plaintext is an English
- The key is derived by a meaningful word

## 3 Overview on PBKDF2

The '-pbkdf2' parameter refers to the **PBKDF2** (Password-Based Key Derivation Function 2) function, used for the key stretching, which makes a possibly weak key, typically a password or passphrase, more secure against a brute-force attack by increasing the resources (time and possibly space) it takes to test each possible key.

It works applying a pseudorandom function, such as hash-based message authentication code (HMAC), to the input password or passphrase along with a salt value and repeats the process many times to produce a derived key, which can then be used as a cryptographic key in subsequent operations.

Having a salt added to the password reduces the ability to use pre-computed hashes (rainbow tables) for attacks, and means that multiple passwords have to be tested individually, not all at once. The US NIST recommends a salt length of 128 bits.

## 4 Build the solution

Following the same approach of the previous homework, I used the MacOS terminal to run the tests and the time command for measures. This time I could not type the command for each operation so I decided to introduce the GNU Bash, a command language for Unix shell.

Given the fact that I did not know the password for the given ciphertext I encrypted a new file with a chosen password and used it for tests with a smaller vocabulary, containing the password.

For the first attempt I remembered the ”bad encryption” error given by the shell using a wrong password, that I noticed last time. I decided then to iterate trough the vocabulary until the error was not appearing, and so try to decrypt with the supposed correct password.

Unfortunately, I discovered soon that this error can just be generated by a different padding and does not reveal anything about the correctness of the generated text.

Rejected the first method I started to look at more complex methods, I found that a possible way to understand which message is the correct one, is by its entropy. This is a good general solution, but at this point I remembered that the plaintext was knowing to be an English text.

This information, apparently, could seems irrelevant, but does actually simplify a lot the identification of the correct decrypted text. In fact, the decryption with a wrong password generates a random text with many different values, the correct one, as an English text, only contains ASCII values.

After all these considerations, I built the following final solution:

```bash
# read each line of the file
while IFS= read -r line; do
  #decrypt and check if the file is composed only by ASCII values
  openssl enc -aes-192-cbc -pbkdf2 -d -in ciphertext.enc -out result.txt -pass pass:${line}
  if file result.txt | grep 'ASCII text'; then
    password=$line
    break
  fi
done < dictionary.txt

#print results on shell
echo "  "
echo "PASSWORD:" $password
echo "To read the decrypted message look at result.txt in the current folder "
echo "  "
```

For the vocabulary I started by using the Unix vocabulary provided by the Unix systems, but after few minutes running, supposing that also the password was in English, I tried with a smaller only English vocabulary (the one in the bibliography). It gave me that the following results:

- Correct password: learning
- Decrypted text: To be, or not to be: that is the question: Whether ’tis nobler in the mind to suffer The slings and arrows of outrageous

fortune, Or to take arms against a sea of troubles, And by opposing end them? To die: to sleep; No more; and by a sleep to say we end The heart-ache and the thousand natural shocks That flesh is heir to, 'tis a consummation Devoutly to be wish'd. To die, to sleep; To sleep: perchance to dream: ay, there's the rub; For in that sleep of death what dreams may come When we have shuffled off this mortal coil, Must give us pause: there's the respect That makes calamity of so long life; For who would bear the whips and scorns of time, The oppressor's wrong, the proud man's contumely, The pangs of despised love, the law's delay, The insolence of office and the spurns That patient merit of the unworthy takes, When he himself might his quietus make With a bare bodkin? who would fardels bear, To grunt and sweat under a weary life, But that the dread of something after death, The undiscover'd country from whose bourn No traveller returns, puzzles the will And makes us rather bear those ills we have Than fly to others that we know not of? Thus conscience does make cowards of us all; And thus the native hue of resolution Is sicklied o'er with the pale cast of thought, And enterprises of great pith and moment With this regard their currents turn awry, And lose the name of action.–Soft you now! The fair Ophelia! Nymph, in thy orisons Be all my sins remember'd.

- execution time: 2,35 minutes

To notice that the execution time depends a lot on the chosen vocabulary and where the password is. Removing the break and iterate trough all the possible entries it needs 6,20 minutes with 10000 words.

## 5 Conclusions

This test reveals how choose a strong randomized password can increase exponentially the confidentiality. Using a very common word, such as one in the 10000 of the smaller vocabulary, introduce a significant weakness, the time required to find the key is quite short. Trying to run the same test on a bigger vocabulary, the time required to evaluate all the possible entries already becomes much higher.

Using a word not included in a vocabulary requires to run a brute force attack, which implies a number of possible combinations so high, that in most case becomes impossible to reach the solution.

## References

[1] Course slides

[2] vocabulary: https://www.mit.edu/ ecprice/wordlist.10000

[3] https://en.wikipedia.org/wiki/PBKDF2