Practicing with an offline dictionary attack

HW2 - CNS Sapienza

Giulio Serra 1904089

November 14, 2019

Contents

1 Introduction

The aim of this essay is to describe how to efficiently perform an offline dictionary attack on an AES encrypted cipher text, using the Open SSL library and Java developing environment.

2 Offline Dictionary Attack

An offline dictionary attack revolvs around using a set of words to attempt to generate potential keys to decrypt a cipher text.
In particular, inside a dictionary, not all words have the same probability to be used in a spoken language(and consequently, as passwords), so to efficiently perfom an offline dictionary attack we should preoritize those and only later going on a full brute force attempt wit a full dictionary.

If we are attempting to break inside a system or decrypting a message, already knowing some of the passwords used in other system/messages by the same users we are trying to attack, we can use them before attempting a brute force attack with a dictionary(because people usually re-use passwords across different systems).

3 Analyzing the Encrypted data

In order to even try to attempt an offline dictionary we first need to gather some informations on how the message was encrypted(the combination of cipher + operating mode), luckily, in this case all this informations were provided to us by the command used for encrypting the plaintext:

```
openssl enc -aes-192-cbc -pbkdf2 -e -in <inputFile.txt> -out ciphertext.enc
```

So, the ciphertext was encrypted AES(Advanced Encryption System, already discussed in the previous essay), in CBC mode with a 192b key using pdkdf2.

3.1 Key Length

The key length of the cipher text is 192b, the average length of dictionary word is 8bit so the the key must have been salted and padded in some way: even opening the cipher text we have the confirm:

```
Salted__... unreadable char
```

3.2 PDKDF2

Upon closer look, in the command used to encrypt the plain text, is included the -pdkdf2 parameter. PBKDF2 stretch a passphrase using a pseudorandom function (such as hash-based message authentication code) along with a salt value, and repeats the process many times(round) to produce a derived key.

3.3 Dictionary

We know that the cipher text contain some text in English language, so we need to start from an English language dictionary to try to discover the key.
The average English dictionary contains around 40.000 words, we need to perform the salt and hash function to each word in order to produce a possible key, so It's quite a lengthy process.
To speed thing up, we can analyze the dictionary to evaluate the most commonly used words.
Luckily google already provide the 1000 more common used words in searches and common spoken language(computed using a machine learning algorithm), so we can first try those words to speed up the attack.

4 Designing the JAVA Implementation

In order to design the implementation of an offline dictionary attack in JAVA we need to design several classes to: parse the complete and partial dictionary (Dictionary.java), to parse the encrypted file (cipherText.java), to decrypt the file (AES.java) and finally to count the elapsed seconds(StopWatch.java).
The algorithm needs not only to output a valid key, but also need to produce a valid plain text (only ASCII characters) because multiple passwords (words) can provide valid keys but only one of them (the real key) will produce a valid plain text.

4.1 AES decryption

In the AES class we need to define a `bruteForceFile(byte[] encrypted)` method that take an array of bytes (rapresenting the cipher text) and tries to decrypt it multiple times, reading a word at a time from the dictionary, until we get a valid output. Here is the implementation:

```
/*
 * Try to brute force an encrypted file with AES
 */
public bruteForceWrapper bruteForceFile(byte[] encrypted) {

StopWatch watch = new StopWatch();
```

```
Dictionary dic = new Dictionary();
bruteForceWrapper wrapper = new bruteForceWrapper(null,0.0);
super.print = false;

System.out.println("Decrypting...");

while(1 == 1) {

if(dic.getWord() == null) {
wrapper.elapsedSeconds = watch.getSeconds();
break;
}

String word = dic.getWord();
encriptionWrapper enc = decript(encrypted,new AESCipherKey(word));

if(enc != null) {
if(enc.isPlaintTextValid()) {
wrapper.data = enc.data;
wrapper.key = word;
wrapper.elapsedSeconds = watch.getSeconds();
break;
}
}

}
super.print = true;
return wrapper;

}
```

The `decript(encrypted,new AESCipherKey(word));` method, need to be tuned accordingly to the considerations explained in paragraph 2, here is the implementation of a decryption in AES using salt and PDKDF2:

```
@Override
public encriptionWrapper decript(byte[]cipherText,AbstractCipherKey key) {

StopWatch timer = new StopWatch();

if(super.print) {
System.out.println("------------------------------------------------------------");
System.out.println("Starting " + this.toString() + " decryption" + " in "
+ opm.toString() + " mode."+ " (" + this.getMode() + ")");

}
```

```
try {


Cipher dcipher = Cipher.getInstance("AES");
AESCipherKey aes_key = (AESCipherKey)key;

SecureRandom random = new SecureRandom();
byte[] salt = new byte[16];
random.nextBytes(salt);

KeySpec spec = new PBEKeySpec(aes_key.getPassword().toCharArray(), salt, 65536, 192); // AES
SecretKeyFactory f = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
byte[] secure_key = f.generateSecret(spec).getEncoded();
SecretKeySpec keySpec = new SecretKeySpec(secure_key, "AES");

            dcipher.init(Cipher.DECRYPT_MODE, keySpec);
            byte[] utf8 = dcipher.doFinal(cipherText);


if(super.print) {

System.out.println("Encryption ended. Elapsed Time: " + timer.getSeconds() + "s");
System.out.println("encrypted message length: " + utf8.length);
System.out.println("-----------------------------------------------------------");

}

return new encriptionWrapper(utf8,timer.getSeconds());


} catch (Exception e) {


if(super.print) {
System.out.println(e);
System.out.println(this.toString() + " decryption failed. \n");
System.out.println("decryption ended. Elapsed Time: " + timer.getSeconds() + "s");
System.out.println("-----------------------------------------------------------\n");
}

return null;
}
}
```

5 JAVA Execution and Results

Here is an example of and execution in JAVA attempting to break the cipher:

```
-----------------------------------------------------------
Starting brute force/dictionary attack:
Decrypting...
10000 left to try
9999 left to try
9998 left to try
9997 left to try
9996 left to try
9995 left to try
9994 left to try
..more..
```

The program transform every word in the dictionary in a possible key in about 10word/s for the small dictionary(commonly used words) and 1word/s for the complete dictionary so we are looking for a total of: 1000s + 48000s (about 13 hours) and still the program most likely will not find the key of the cipher, that's because of this lines of code:

```
...
KeySpec spec = new PBEKeySpec(aes_key.getPassword().toCharArray(),
salt, 65536, 192); // AES-192b
SecretKeyFactory f = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
...
```

When attempting to decrypt, the Java Cipher library force us to specify the number of rounds and dimension of the SALT (two informations we don't have) to generate a key to try to decrypt our cipher.
So to write a program that can actually brute force our cipher, we have to try every possible combinations of salt/rounds/words.

6 OpenSSL Library

Looking once again to the provided informations, the command used to encrypt the cipher text, belongs to a library for unix operating systems.It's possible to write a more efficient implementation of the same algorithm used in JAVA, just using a shell script with the default parameters of decrypting using AES and pbkdf2.

6.1 Implementation

So, once again, we need to try all the most commonly used words in the english dictionary, generate a valid key and check if the decryption output a valid plain text(only ASCII characters), here is the implementation:

```
#!/bin/bash

COMPLETE_DICT='/Users/giulioserra/Documents/Universita/CNS2/englishDic.txt'
FREQUENT_DICT='/Users/giulioserra/Documents/Universita/CNS2/commonWords.txt'
OUTPUT='/Users/giulioserra/Documents/Universita/CNS2/plain.txt'
COMMONWORD=false;

echo "Starting AES bruteForce attack..."
start=$SECONDS

while IFS= read -r LINE
do
  DECRIPTION=$(openssl enc -aes-192-cbc -pbkdf2 -d -in ciphertext.enc -out $OUTPUT -k $LINE
    if [ "$DECRIPTION" != "bad decrypt" ]
        then
            PLAIN=$(<$OUTPUT)
            if [[ $PLAIN == *[![:ascii]]* ]];
            then
                echo $LINE;
            else
                end=$SECONDS
                COMMONWORD=true;
                duration=$(( end - start ))
                echo "file decrypted, password: $LINE , elapsed time: $duration s";
                exit
            fi

    fi

done < "$FREQUENT_DICT"

if [ !"$COMMONWORD" ]
... same as before but with complete dictionary...
 fi
```

7 Final Results

Here is an execution of the script:

```
(base) MBP-di-Giulio:CNS2 giulioserra$ sudo /Users/giulioserra/
Documents/Universita/CNS2/decryptor
Starting AES bruteForce attack...
out
team
offers
side
```

```
file decrypted, password: learning , elapsed time: 12 s
(base) MBP-di-Giulio:CNS2 giulioserra$
```

Luckily the key was in the commonly used words dictionary, so the script is able to clock the decryption of the file in just 12 seconds, here is the plain text using the word "learning":

```
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.
```

8 References

Dictionary of the commonly used words according to google:
https://github.com/first20hours/google-10000-english/blob/master/google-10000-english.txt
The OpenSSL Library:
http://www.wolfssl.com/?gclid=EAIaIQobChMIuvnJkrbj5QIV0uR3Ch3fJw1NEAAYASAAEgLFyPD_BwE
Java Cipher Class:
https://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html