Universidad Carlos III de Madrid

Security Group – Dept. of Computer Science

Universidad Carlos III de Madrid

Access Control and Authentication

Security Engineering
Fall 2011

Part I: Passwords

Goal:

- Learn how to appropriately choose and manage passwords.

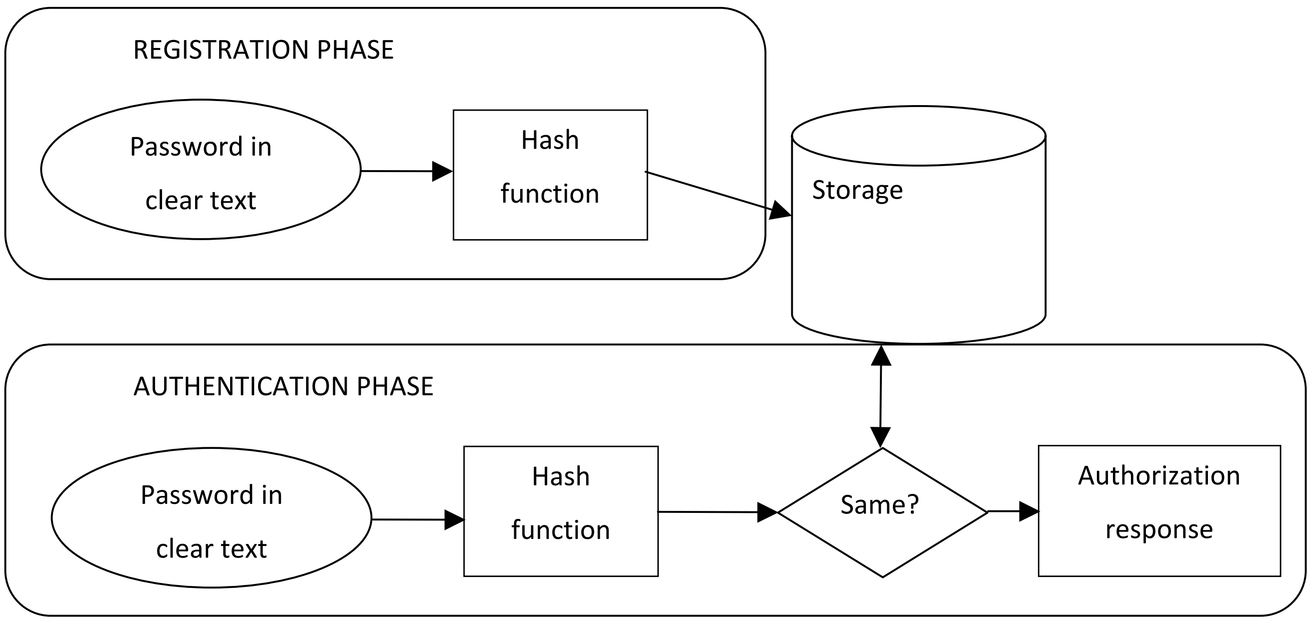
Tool:

John the ripper. Download it from http://www.openwall.com/john/

Introduction

Passwords are one of the most commonly used methods to authenticate users. Choosing a *good* password is crucial to provide some security guarantees, in particular to prevent an attacker from guessing it and thus get access to the user's privileges. Security administrators must also have an appropriate security policy as far as password management is concerned.

In the classical password-based authentication setting, the user is first asked to make a claim about his identity (e.g., by providing a user name). To prove that he is really who he claims to be, he then must introduce a secret password that, ideally, is only known by him. Upon verifying that the data provided coincides with that stored in the system, the user is granted access. One critical issue in this scheme is that the device queried by the verifier to retrieve the user's password can be compromised. This is generally solved by not storing the passwords *in clear text* anywhere. Instead, a digest of the password (computed by applying a secure hash function to it) is kept within a file with restricted access. This does not change substantially the verification process: the digest of the provided password is checked against the digest stored in the system. The process is illustrated in the following chart:



An attacker who gets access to the password file will not be able to learn them directly, as only the digests are stored and the (ideally!) one-way nature of the hash function guarantees that the preimage cannot be obtained. However, nothing prevents the attacker from making a guessing about the password, compute its digest, and then check whether it coincides with the one stored in the file. Without any other prior knowledge, the attacker must try every possible candidate password, in whatever order he chooses, until the correct one is found. This is known as a *brute force attack*.

Fortunately for the attacker, the process can be improved thanks to a simple fact: Users tend to choose passwords that are easy to remember. This reduces considerably the space of possible passwords to be tried, for example by using combinations of words from the user's native language. A common way of carrying out this attack is by using a *dictionary* containing a large number of "words" that could appear somewhere within a password. Candidate passwords are then generated by trying different combinations of such words with various lengths. Even though the number of possible passwords is still huge, the dictionary attack gives a considerable speed-up to the attacker.

Dictionary attacks can be prevented by choosing *good* passwords. These should not be short (no less than 8 characters) and must contain numbers, letters (both upper- and lower-case) and punctuation symbols. Furthermore, neither the entire password nor any substring should have any relation with information that can be deducted from the user's identity, such as his name, relatives, relevant dates (e.g. year of birth), hobbies, preferences, etc.

In the first part of this practical we will carry out a dictionary attack against a password file. This will give a measure of the strength of different choosings.

Exercise 1: Software setup

Download *John the ripper* from the URL given above and the material (a rar file containing a password file and a dictionary) available at AG2. Once the program is extracted, there should be a folder named *run*. Extract the password file and the dictionary and put them in this folder.

To compile the program:

1 Go to the *src* folder

2 Execute *make clean generic*

If the compilation goes OK, an executable is generated in the *run* folder.

Exercise 2: Guess the weakest passwords

Execute *John* and have a look at the different options. We will first run a simple search. By using the –*show* command we can see that *John* remembers those passwords recovered in previous searchs, so they are not attacked anymore.

Exercise 3: Basic dictionary attack

Execute *John* again without any modifier. By default, *John* comes with a small dictionary to be used (password.lst) if none is provided. We will later use the dictionary provided with the practical. (NOTE: the password file was generated by Spanish users, so the dictionary is based on Spanish words.) As an initial attempt, execute an attack using the dictionary by default (password.lst)

Exercise 4: Advanced dictionary attack

Passwords are often combinations of common words. *John* can try these combinations by specifying rules in the configuration file (*john.conf*). The format of such rules is described at http://www.openwall.com/john/doc/RULES.shtml. Open the configuration file and find the section "*Wordlist mode rules*". Here you will find the predefined rules and some illustrative comments. We will add a new simple rule specifying that plurals of Spanish words must be tried as well. This is done by adding es at the end and then executing *John* with the option --*rules*.

Exercise 5: Brute force attack (with dictionary)

If the use of rules fails, we are left with the only option of attempting a dictionary-based brute force attack. *John* does this through the so-called incremental mode (option -i), which tries all possible combinations of characters contained in words, starting with those of shorter lengths. Before starting with any character, *John* will try digits (option –i:Digits).

After this, try also a general brute force attack with the -i option.

Part 2: Digital Signatures and E-mail

Theory

To be revised in the Lab during the practical. Main points:

- Digital signatures: functions and properties
- Cryptographic elements
- Digital certificates
- Signing and verifying messages

Exercise 0: Generating self-signed certificates with Acrobat Reader

Download Acrobat Reader if it is not already installed on your computer. Hava also a look at: http://help.adobe.com/en_US/Reader/8.0/help.html?content=WSAC8084C2-14F7-4841-9EF8-92106D22C3DB.html [Check: "Adding digital signatures to PDFs -> QuickStart: Create a self-signed certificate]

Exercise 1: Visualising fields in the certificate

Note: If your certificate is in pkcs12 format (extension .p12), convert it to PEM format with:

```
openssl pkcs12 -in <CERT>.p12 -out <CERT>.pem -clcert
```

1.1. Check the issuer

```
openssl x509 -noout -in <CERT>.pem -issuer
```

1.2. Check the owner (subject)

```
openssl x509 -noout -in <CERT>.pem -subject
```

1.3. Find out the dates

```
openssl x509 -noout -in <CERT>.pem -dates
```

1.4. View all the fields

```
openssl x509 -noout -in <CERT>.pem -text
```

Exercise 2: Signing emails in Mozilla Thunderbird

1. Download and install Mozilla Thunderbird

   Get it from http://www.mozilla.org/es-ES/thunderbird/ and unzip it within a folder (tar –xjvf <file_name>). Execute binary file *thunderbird* located in the *run* folder.

2. Setup an account.

   Due to some access restrictions, you **cannot** user your university's email account for this process. You can use either your departamental account (the one used in the Lab) or any other personal account (gmail, hotmail, etc.) Go to Edit/Configuration and follow the instructions.

3. Instal your personal certificate and the CA's certificate.

   ```
   Open Mozilla Thunderbird

   Edit / Configure accounts / Security

   View Certificates / Authorities / Import

   View Certificates / Your Certificates / Import

   –    Trust this CA to identify email users

   –    View certificate: check content
   ```

4. Configure your account.

   ```
   Edit / Configure accounts / Security

   Digital signatures / Choose (certificate to sign)
   ```

5. Send a signed certificate

```
          Write some email

        Security / Digitally sign this message
```

Exercise 3: Signing PDFs with Acrobat Reader

Familiarise yourself with the configuration and basic usage of certificates in Acrobat Reader.

Try also to verify signed documents.