PRACTICE 1: PASSWORDS

PREVIOUS KNOWLEDGE

To do this practice it is necessary to have a basic knowledge about symmetric key cryptography and about the user authentication mechanisms used in UNIX and Windows systems.

INTRODUCTION

A "good" selection of passwords by the users of a system is highly important to maintain security on those systems with no or only weak access restrictions and with high network connectivity, for example, those systems which are connected to the Internet.

The objective of all attackers usually is, in general, to gain privileges of *root* (i.e., the administrator), to get full access to the system resources. Clearly, this is relatively difficult to achieve from outside a system. However, if the attack is made from inside the system (i.e., from a non-root user account), the task is much simpler for the attacker.

For that reason, the first step of an attacker is normally to try to gain one user password, so that the attacker can access the system and run a second subsequent attack on the root password from inside. From the several possible forms to do so, the most usual ones (in UNIX systems) are dealing with the "`passwd`" file, which is usually located in the directory "`/etc/`". Among other information, all user passwords are stored in this file. Normally, this file is "shadowed", that means, hidden under root privileges, so that it cannot be read easily. The techniques that attackers use to access this kind of hidden files differ from system to system and are not the objective of this practice. We assume that this file can be accessed by the attacker.

The idea of this practice is to start from a file with encrypted passwords, such as the file "`/etc/passwd`" (possibly obtained after unhiding a "shadowed" file). From this file, user's passwords shall be obtained which shall make use of both, any available knowledge about the users, and also knowledge about the common rules which users apply very often when they choose a password. We will also examine in detail the aspects that influence the performance when trying to obtain one or several user passwords in this way, since this performance is important for the security of the system. In addition, we will also study the authentication scheme that is used in the Windows NT/2000 operating systems. Finally, we will study some techniques that are used to discover user passwords when these passwords are interchanged through a communications network (password sniffing).

PRINCIPLES

The format of password files, which are used in this first practice, can vary from one operating system to another. However, the basic format of a single line in the password file is the same:

```
Login : Password : ID : Group : GECOS : User Directory : Shell
```

Two fields are interesting here: The "`login`" name of a user, which is the first field of a line, and the "`password`" field directly afterwards, both separated by a colon. We will consider systems where the creation of the password field is based on standard DES encryption, as explained in the theoretical sessions, although there are also further schemes based on other techniques such as MD5 and the like.

This practice will make use of the program "John the Ripper v1.7" (http://www.openwall.com/john/), which is a tool for cracking passwords and which is currently available for UNIX, DOS, and Windows 95/NT systems. Its basic intention is to detect weak UNIX password in a system (hopefully before an attacker does…)

PRACTICE AND QUESTIONS

John the Ripper is a command-line application, which is why we have to use it from the command prompt available in Windows. The program executable is called "`john`" and the different command-line options will be explained in the following when necessary. For convenience, it is recommended to copy the passwords files, which shall be examined by "`john`", into the directory with this program executable.

`john` operates on all passwords in a file simultaneously (unless the opposite is requested by a command-line option). When it discovers the password of one user, it is stored in the file "`john.pot`". The login of this user is not processed further to increase the efficiency of the cracking process. In order to increase the efficiency even further, the '`-u: user`' option might be of interest, which allows for restricting the cracking process to a single user account, increasing the number of cracking operations per second on this password. Thus, if we suspect that the password of a certain user is associated with a certain dictionary, it might be useful to try to crack only the password of this user in order to save time.

Everything that we have just explained, might be useful for cracking passwords in the following part I in order to eliminate the large number of possible simple passwords. After this first step, the complexity of how to crack passwords might be increased as a second step. Nevertheless, proceeding in these steps does not "constitute a standard". It is left to the skills and the experience of everyone what to do first or last.

NOTE:

- You need to run this practice using the Windows operating system.
- A Zip file is provided with this practice (1P_Passwords_pack.zip), containing all the files and programs that are necessary to do the practice.
- For this practice, use the version of the program '*John the Ripper*' that is provided in the complementary material.
- It is suggested to consult the enclosed documentation for the program '*John the Ripper*' before starting to prepare the practice.

Obtaining passwords

In this part, as many passwords as possible shall be obtained from the password file "`passwords1.txt`". For this reason, the different available execution modes of the program '`john`' shall be used to realize a series of common attacks based on dictionaries, which have be coded systematically. The obtained results shall be compared with the encrypted passwords in the password file. Furthermore, a cracking application shall be programmed, so that certain rules on the 'morphology' of the words in the dictionary can be applied mechanically (e.g., inverting characters, insertion or substitution of characters etc.). This is intended to detect the most common modifications users make to the chosen password. (Normally, users tend to choose a well-known pattern, which is easy to remember, and make some modifications so that the chosen password is not too easy to crack.) Finally, the complexity of a brute-force attack on the same password file shall be evaluated.

- At first, the command-line option "`-single`" shall be tried for convenience. This option executes the program '*John the Ripper'* in the mode "Single Crack", which allows to run a set of simple test methods:

```
john-mmx -single passwords1.txt
```

1. **Which passwords can be obtained using this option? Examine the enclosed documentation for the program '*John the Ripper*' and respond in detail to the following questions:**

2. **Indicate which methods the 'Single Crack' mode comprises in general. Which concrete test methods were used to obtain the passwords (which you have found as solution to the first part of this question), from the password file '`password1.txt`'?**

- The next step is to try a dictionary attack using those dictionaries which are provided in the complementary material for the practice. Therefore, *John the Ripper* is to be executed in the mode "wordlist" using the command-line option "`-wordlist`". For the following question, use an English dictionary ("`english.txt`") and a further one containing words related to the cinema ("`cinema.txt`"):

```
john-mmx -wordlist=dictionary-file passwords1.txt
```

3. **Indicate for each dictionary the set of obtained passwords. What is the relation between the size of the dictionary and the time it takes to make a complete test?**

- After having found some passwords, obtained directly from the dictionary, the following phase of the practice will consist of applying certain variations on the dictionary words before encrypting them. For example, words can be reversed or certain characters can be inserted or exchanged etc. These variations can be programmed using a simple language, but in this section we will use the predefined variations in the configuration file of '`john`' ("`john.ini`"). In order to activate these default variations, the command-line option "`-rules`" is used:

```
john-mmx -wordlist=dictionary-file -rules passwords1.txt
```

4. **For each dictionary, indicate the passwords you have found with this new option as well as the rules used for obtaining them. You can locate the rules in the file "`john.ini`" and consult the enclosed documentation to *John the Ripper*.**

- After having used these tests with the basic rules, you are now required to program some new rules on your own. Although the rules often depend on the language or the computer science culture, there are some changes that are almost universally used: For example, Changing "a" by "@" or "&", "o" by "0", "c"s by "(", "k" by "q", "i" by "!", to add "." or to add "#" in the end or at the beginning of the words, etc. It can also be useful in some cases to eliminate the rules that already have been used in order to increase the performance of the tests.

5. **Program new rules which allows for obtaining as many passwords as possible from the password file. Indicate the obtained passwords and the set of rules which were involved in getting the password.**

- In case all the previous tests fail, `john` allows for doing a **brute-force** attack. For this purpose, it can be executed in the "Incremental" mode using the "`-incremental`" command-line option. This mode is the most powerful one implemented in *John the Ripper*: It allows for trying all possible combinations of characters as passwords. This incremental attack must be configured in the file "`john.ini`", in order to select the characters from which the combinations shall be created (it might be known, for example, that the password only consists of numbers). Furthermore, the minimum length and maximum length of the password to be obtained can be configured in this file, as well (for example, in case you assume that the password is a DNI or a telephone number). `john` provides various pre-defined incremental modes ('All', 'Alpha', or 'Digits') in the file '`john.ini`' which can be used in this practice.

6. **Knowing that the password of the user '`grace`' is a numerical password of 8 digits, design and execute an incremental attack which obtains this password using ONLY those passwords consisting of 8 numerical digits. In addition to the obtained password, show in detail the changes, you have made to the file '`john.ini`'.**

   NOTE: It is suggested that you configure and use the incremental mode 'digits', which utilizes only numerical digits for the generation of the passwords, which shall be compared to the one to obtain. Similarly, to avoid unnecessary tests, it is recommended to focus the process of searching the password to the user '`grace`' using the command-line option '`-users`'.

- Finally, `john` enables a user to implement an own algorithm for the generation of passwords to be compared to the ones in the passwords file. For this purpose, it is necessary to define and execute the mode "External". The definition of the algorithm is made in the configuration file, in which the user must implement those functions that `john` should use for the generation of passwords. This is done using a subset of the C programming language.

- Suppose for the sake of security, that the user '`ally`' changes her password monthly. The selected password is always according to the following pattern:
  - It is a password with six characters
  - The four first characters are letters from the alphabet (capitalized and/or non-capitalized): {a, b, c, …, z, A, B, C, … , Z}
  - The two last characters represent the current month as numerical values {01, 02, …, 12}

7. **Define and execute an external mode which allows for obtaining the password of the user '`ally`'. Show the obtained password and the changes made to the configuration file '`john.ini`'.**

8. **Execute a generic incremental attack (incremental mode 'All') on ONE of the remaining passwords, which have not been discovered up to now. Is it reasonable to execute such an attack on the set of all passwords which have not been discovered up to now?**

Part II
The influence of SALT onto the process of obtaining passwords

In this part, a number of question is raised which will put your attention to the relevance of the SALT in the process of obtaining passwords and its impact on the security of a UNIX system.

9. **Based on the knowledge that we have on the model of generation and verification of keys (DES, SALT, etc.) explain analytically if it is possible that:**
   a) **Two users with different logins have the same key.**
   b) **Two users with the same login have different keys.**
   c) **Two users have the same SALT.**

10. **Explain analytically what is the exact meaning of the phrase "loaded 41 passwords with 37 different salts". From the point of view of the security of the keys, indicate what are the implications of saying "loaded 41 passwords with 1 different salt".**

- Suppose that five users have an account on a UNIX system. These five users have chosen numerical passwords with 7 digits. Moreover, the password of each user is the same on two different systems. Assume that you have a copy of the password file from these two UNIX machines, "`password2.txt`" and "`password3.txt`", which contain only information about these five users. Execute an incremental attack separately for each of these two password files, using the following commands:

  ```
  john-mmx -incremental=digits passwords2.txt
  john-mmx -incremental=digits passwords3.txt
  ```

  NOTE: As in the previous case, modify the corresponding parameters of the incremental mode 'digits' for the execution of these two commands, so that the keyspace is limited only to the set of numerical passwords having 7 digits.

11. **Looking at the results obtained from the two commands, answer in detail to the following questions:**
    a) **Compare the obtained passwords from both files. What can you observe?**
    b) **Compare the execution time of `john` for both files. What can you observe?**
    c) **Are the obtained results logical? Why?**

12. **If a SALT with 18 bits (3 characters with 6 bits each) is used, so that the length of the corresponding ciphertext remains constant (11 characters a 6 bits), evaluate the following parameters:**
    a) **The effectiveness of an attack with *John the Ripper***
    b) **The efficiency of an attack with *John the Ripper***
    c) **The security of the system against an attack using precomputed dictionaries.**