A Turkish Password Cracker for UNIX Based Operating Systems

By

Osman Okyar TAHAOĞLU

A Dissertation Submitted to the
Graduate School in Partial Fulfillment of the
Requirements for the Degree of

MASTER of SCIENCE

Department: Computer Engineering
Major: Computer Software

İzmir Institute of Technology
İzmir, Turkey

June, 2001

We approve the thesis of **Osman Okyar TAHAOĞLU**

Date of Signature

.....

27.06.2001

Assoc. Prof. Dr. Ahmet H. KOLTUKSUZ
Supervisor
İzmir Institute of Technology
Department of Computer Engineering

.....

27.06.2001

Prof. Dr. Şaban EREN
Ege University
Department of Computer Engineering

.....

27.06.2001

Asst. Prof. Dr. Tuğkan TUĞLULAR
İzmir Institute of Technology
Department of Computer Engineering

.....

27.06.2001

Prof. Dr. Sıtkı AYTAÇ
İzmir Institute of Technology
Head of Department

ACKNOWLEDGEMENTS

I would like to thank to my advisor *Assoc.Prof. Ahmet H. KOLTUKSUZ, Ph.D.,* firstly for his advice about studying on this subject and for his enduring support and supervision that made this thesis possible.

Additionally, my deepest thanks to my family, to the academic staff of my department, and to my colleagues for their support and encouragement.

ABSTRACT

UNIX and UNIX-based operating systems have been widely utilized in local area and wide area network systems that supply application and development chain of users through remote access as well as online connections. Although UNIX operating system has got powerful tools which have secure methods for user authentication, user management and for password storage, weak password choices of the users affect the entire system security negatively.

This study aims to crack the password hashes which are encrypted by DES using the method of dictionary attack. The developed application is introduced and compared with the previous utilities. Password encryption, password storage, and the structure which was developed against a possible dictionary attack of UNIX are examined. The good password choosing method for the users and the system administrator are given.

ÖZ

Kullanıcılara uzaktan erişim, uygulama ve geliştirme sağlayan pek çok yerel ve geniş bilgisayar sistemlerinde UNIX ve UNIX tabanlı işletim sistemleri kullanılmaktadır. Unix tabanlı işletim sistemlerinde, kullanıcıları tanımlama ve yetkilendirme araçları güçlü şifreleme ve saklama tekniklerine sahip olmalarına karşın, kullanıcıların zayıf olarak tanımlanabilecek şifre seçimleri, kullanıcı yönetimi ve güvenliğini olumsuz yönde etkilemektedir.

Bu çalışmayla, DES algoritması ile şifrelenmiş kullanıcı şifrelerinin sözlükten tarama ve saldırı yöntemiyle kırılması amaçlanmış ve geliştirilen uygulama ile daha önce aynı amaçla yazılmış programlar karşılaştırılmıştır. İşletim sisteminin şifreleme yöntemi, kullanıcı adlarını ve şifrelerini saklama yöntemi ve olası sözlükten saldırılara karşı geliştirdiği yapı incelemiştir. Kullanıcılara ve sistem yöneticilerine iyi şifre seçim tanımı yapılmıştır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZ v
TABLE OFCONTENTS vi
LIST OF FIGURES ix
LIST OF TABLES x
CHAPTER 1 INTRODUCTION 1
1.1 Motivation 1
1.2 Objective 1
1.3 Scope and Structure 1
CHAPTER 2 CRYPTANALYSIS AND DES 3
2.1 History 3
2.2 Standards for the Modes of DES 4
2.3 Data Encryption Algorithm 4
2.3.1 Introduction 4
2.3.2 Enciphering 4
2.3.3 Deciphering 7
CHAPTER 3 PASSWORD STRUCTURE IN UNIX 11
3.1 Introduction 11
3.2 User Authentication 11
3.2.1 Users and Passwords 11
3.2.2 Password for System Accounts 11
3.3 Password Security 12
3.3.1 /etc/passwd File 12
3.3.1.1 User Name 13
3.3.1.2 Understanding Default System User Names 13
3.3.1.3 Password 14
3.3.2 Salted Passwords 15
3.3.2.1 The Threat of the DES Chip 15
3.3.3 Login Procedure 16
3.3.4 crypt( ) 17
3.3.5 Shadow File Entry 17

CHAPTER 4 VARIOUS ATTACK SCHEMES ON PASSWORD..... 19
4.1 Introduction 19
4.2 The Importance of Good Passwords..... 19
4.3 Human Password Choices and System Security..... 20
4.4 Password Attacks..... 20
4.4.1 Dictionary Attacks..... 21
4.4.2 Fast Crypt Implementations..... 21
4.4.3 Precomputed Encrypted Dictionaries..... 22
4.5 Automated Dictionary Attack..... 22
4.5.1 Known Encryption Algorithm..... 23
4.5.2 Acceptable Running Times..... 23
4.5.3 Encrypted Password Availability..... 23
4.5.4 Decreasing Password Guessability..... 23
4.5.5 Other Approaches..... 24
4.6 Password Composition Vulnerabilities..... 25
4.6.1 Bad Passwords..... 25
4.6.1.1 What Not to Use as a Password 26
4.6.2 Good Passwords 27
4.6.2.1 What to Use..... 27
4.6.2.2 Reducing Break-in Possibilities..... 28
4.6.2.3 Password Screening..... 28
4.6.2.4 Picking Good Passwords..... 29
4.6.2.4.1 Method to Choose Secure and Easy to Remember Passwords..... 29
4.6.2.5 Improving Passwords Against Social Engineering..... 30
4.7 Case Studies: Cracking Linux Passwords via Dictionary Attack..... 31
4.7.1 L0phtCrack 2.5..... 31
4.7.2 John the Ripper 34
4.7.3 Crack..... 35
4.7.4 Other Linux-Compatible Password Auditing Tools..... 36
CHAPTER 5 TURKEY TURKISH BASED PASSWORD CRACKER..... 38
5.1 Design of the Utility..... 38
5.2 Selection of the Medium and Tools..... 38

5.3 Implementation of the Program..... 39
5.4 Results..... 42
CHAPTER 6 CONCLUSIONS..... 44
6.1 Concluding Remarks..... 44
6.2 Suggestions for Future Work..... 44
SUMMARY..... 45
ÖZET..... 46
BIBLIOGRAPHY..... 47

LIST OF FIGURES

Figure 2.1 DES Algorithm, Enciphering Computation 5
Figure 2.2 Calculation of f(R, K)..... 8
Figure 3.1 /etc/passwd File 12
Figure 5.1 Salt Generation..... 39
Figure 5.2 Password Cracking Process 40
Figure 5.3 Call to crypt() 41

LIST OF TABLES

Table 2.1 Initial Permutation..... 5
Table 2.2 Inverse Initial..... 6
Table 2.3 E Bit-Selection..... 8
Table 2.4 S1 Function..... 9
Table 2.5 Permutation Function..... 9
Table 3.1 Excerpts from a sample /etc/shadow file from an SVR4 system..... 18
Table 5.1 Hardware Profile..... 38
Table 5.2 Software Profile..... 38
Table 5.3 Results of Experiment #1 42
Table 5.4 Results of Experiment #2..... 43

CHAPTER 1

INTRODUCTION

1.1 Motivation

As computers get more and more into our life and business, security gets more importance. Most medium to large-sized computers and even the smaller computers are now shared among users. Therefore, remote access to computer systems need to be managed under the perspective of security policy.

Even though UNIX operating system is very much security conscious and many of its processes such as authentication, adding, deleting and management of the users have been standardized and more over passwords are kept in a shadowed file in an encrypted format while encryption algorithm(DES) which is used in due process is away from being decrypted however, users' password choices weaken the entire system just like a weak ring in a chain. Uneducated and/or careless users prefer easy to remember passwords for themselves. But the point is a good password security is the first line of defense against the system abuse.

1.2 Objective

This work covers password guessing of UNIX/UNIX-based operating systems by the method of a Turkish dictionary attack. The application program coded in C is executed for a real environment password file on a Linux (SUSE) operating system. Before the case study on cracking the passwords, it is assumed that by somehow the password file (or the shadow file if shadowing is activated) with the known encryption algorithm is accessed. An acceptable running time (and CPU time) and a large on-line Turkish dictionary is available. The percentage of successful guesses relies on the passwords with a significant probability of being in the word list.

Several other crackers which are available on Internet have been examined and compared with the developed program.

1.3 Scope and Structure

In Chapter 1, a brief introduction is made about aim, goal and objectives of the thesis. Chapter 2 deals with the most popular cipher in history, Data Encryption Standard (DES). Describes the modes of the encryption algorithm, examines a brief explanation of the encryption and the decryption procedures.

In Chapter 3, password security in UNIX and UNIX-based Operating Systems is focused. The relationship between the users and the user authentication is examined. Password protection mechanism provided and maintained by the operating system plus the DES algorithm is introduced.

In Chapter 4, the reason for choosing the dictionary attack against user passwords is introduced. The good and the bad password identifications are determined. Advices for the

password choosing are given. Possible dictionary attack methods and the other available password crackers are introduced.

Chapter 5 is the case study of this thesis. It introduces a password cracker coded in C. Constraints about the cracker software which is developed under the circumstances of predetermined realities about the users' weaknesses and the environment are explained in this chapter.

Chapter 6 is a general conclusion involving password cracking via dictionary attack with the method defined in Chapter 4 and with the result in Chapter 5. Necessary remarks and advices are given for the future studies.

CHAPTER 2

CRYPTANALYSIS AND DES

2.1 History

The Data Encryption Standard (DES) is the most popular cipher in history, even though it's been around for the last 25 years.

Because of the unavailability of general cryptographic technology outside the national security arena, and because of various security provisions, including encryption, National Bureau of Standards (NBS) initiated a computer security program in 1973 which included the development of a standard for computer data encryption. Since Federal standards impact on the private sector, NBS solicited the interest and cooperation of industry and user communities in this work. The criteria specified for the call were:[1]

- Provide a high level of security.
- Must be specified and easy to understand.
- Must provide the security independent of the secrecy of the algorithm.
- Must be available to all users.
- Must be adaptable for use in diverse applications.
- Must be economical to implement in electronic devices.
- Must be efficient to use.
- Must be able to be validated.
- Must be exportable.

Many companies developed proposals, but International Business Machines (IBM) prevailed. IBM's DES was subjected to rigorous testing and, by 1977, the NBS and the National Security Agency (NSA) endorsed it. Since then, DES has been the de facto encryption algorithm used in non-classified environments and UNIX/Linux passwords.[2]

Federal Processing Standards Publication 46-2 concisely describes DES as:

"...a mathematical algorithm for encrypting (enciphering) and decrypting (deciphering) binary coded information. Encrypting data converts it to an unintelligible form called cipher. Decrypting cipher converts the data back to its original form, called plain-text."

Both encryption and decryption functions rely on a key, without which unauthorized users can-not decrypt a DES-encrypted message. This key derived from the user's typed password, consists of 64 binary digits. 56 bits are used in encryption,

---

[1] Charles P.Pfleeger. *Security in Computing*, p. 106.

[2] Anonymous, *Maximum Linux Security*, p. 126.

and 8 are used in error checking. The total number of possible keys is therefore quite high.

2.2 Standards for the Modes of DES

The Federal Data Encryption Standard described in Federal Information Processing Standards (FIPS 46) specifies a cryptographic algorithm to be used for the cryptographic protection of sensitive, but unclassified, computer data. This FIPS defines four modes of operation for the DES which may be used in a wide variety of applications. The modes specify how data will be encrypted (cryptographically protected) and decrypted (returned to original form). The modes included in this standard are the Electronic Codebook (ECB) mode, the Cipher Block Chaining (CBC) mode, the Cipher Feedback (CFB) mode, and the Output Feedback (OFB) mode.[3]

2.3 Data Encryption Algorithm

2.3.1 Introduction

The algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key.[4] Deciphering must be accomplished by using the same key as for enciphering, but with the schedule of addressing the key bits altered so that the deciphering process is the reverse of the enciphering process. A block to be enciphered is subjected to an initial permutation ***IP***, then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation ***IP*** **-1**. The key-dependent computation can be simply defined in terms of a function ***f***, called the cipher function, and a function ***KS***, called the key schedule. A description of the computation is given first, along with details as to how the algorithm is used for encipherment. Next, the use of the algorithm for decipherment is described. Finally, a definition of the cipher function ***f*** is given in terms of primitive functions which are called the selection functions ***Si*** and the permutation function ***P***.

The following notation is convenient: Given two blocks ***L*** and ***R*** of bits, ***LR*** denotes the block consisting of the bits of ***L*** followed by the bits of ***R***. Since concatenation is associative, ***B1B2...B8***, for example, denotes the block consisting of the bits of ***B1*** followed by the bits of ***B2***...followed by the bits of ***B8***.

2.3.2 Enciphering

A sketch of the enciphering computation is given in Figure 2.1. The 64 bits of the input block to be enciphered are first subjected to the following permutation, called the initial permutation ***IP.*** (see Table 2.1)

---

[3] FIPS *PUB 81.*

[4] FIPS *PUB 46-3.*

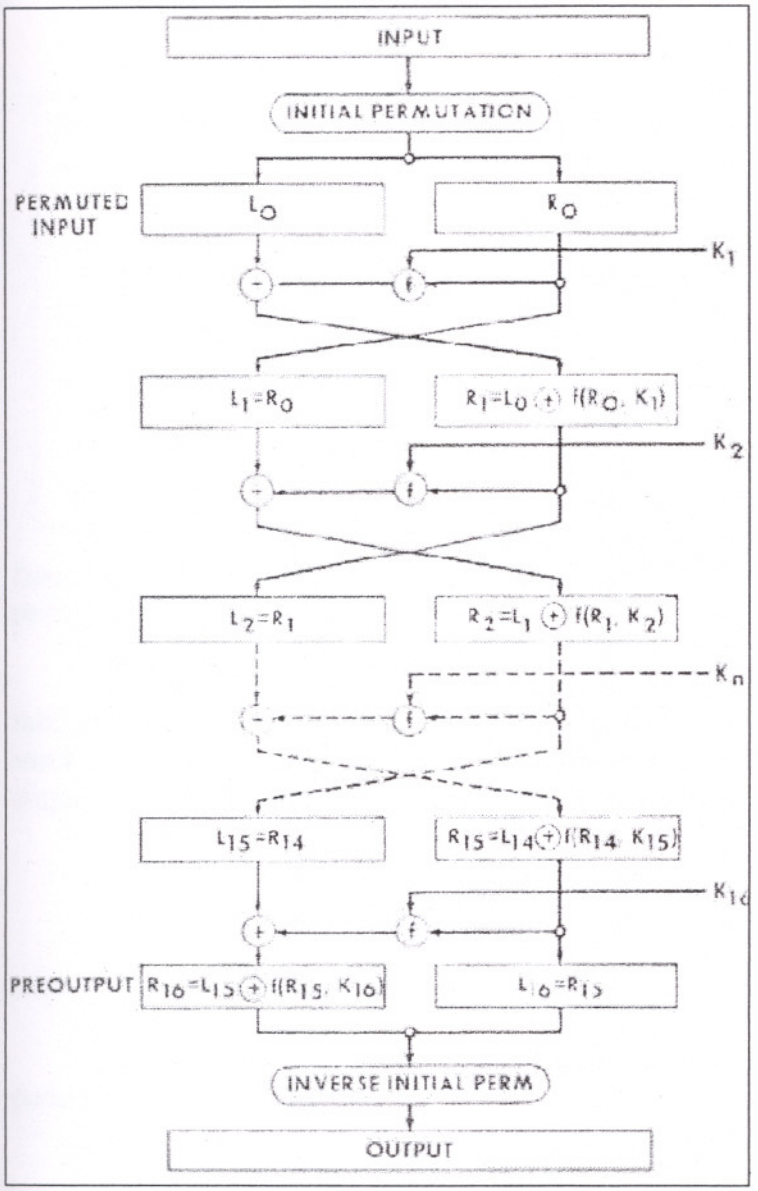

Figure 2.1 DES Algorithm, Enciphering Computation.

Table 2.1 Initial Permutation.

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	48	39	31	23	15	7

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The permuted input block is then the input to a complex key-dependent computation described below. The output of that computation, called the preoutput, is then subjected to the following permutation which is the inverse of the initial permutation as in Table 2.2.

Table 2.2 Inverse Initial.

IP-1							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

The computation which uses the permuted input block as its input to produce the preoutput block consists, but for a final interchange of blocks, of 16 iterations of a calculation that is described below in terms of the cipher function f which operates on two blocks, one of 32 bits and one of 48 bits, and produces a block of 32 bits.

Let the 64 bits of the input block to an iteration consist of a 32 bit block L followed by a 32 bit block R. Using the notation defined in the introduction, the input block is then LR. Let K be a block of 48 bits chosen from the 64-bit key. Then the output $L'R'$ of an iteration with input LR is defined by:

$$L' = R \qquad (1)$$
$$R' = L \oplus f(R,K)$$

where $\oplus$ denotes bit-by-bit addition modulo 2.

As remarked before, the input of the first iteration of the calculation is the permuted input block.

If $L'R'$ is the output of the 16^{th} iteration then $R'L'$ is the preoutput block. At each iteration a different block K of key bits is chosen from the 64-bit key designated by KEY. With more notation we can describe the iterations of the computation in more detail. Let KS be a function which takes an integer n in the range from 1 to 16 and a 64-bit block KEY as input and yields as output a 48-bit block Kn which is a permuted selection of bits from KEY. That is

$$Kn = KS(n,KEY) \qquad (2)$$

with Kn determined by the bits in 48 distinct bit positions of KEY. KS is called the key schedule because the block K used in the n'th iteration of (1) is the block Kn determined by (2).

As before, let the permuted input block be LR. Finally, let $L0$ and $R0$ be respectively L and R and let Ln and Rn be respectively L' and R' of (1) when L and R are respectively $Ln-1$ and $Rn-1$ and K is Kn; that is, when n is in the range from 1 to 16,

$$Ln = Rn-1 \qquad (3)$$
$$Rn = Ln-1 \oplus f(Rn-1,Kn)$$

The preoutput block is then ***R16L16***.

The key schedule produces the 16 ***Kn*** which are required for the algorithm.

2.3.2 Deciphering

The permutation ***IP -1*** applied to the preoutput block is the inverse of the initial permutation ***IP*** applied to the input. Further, from (1) it follows that:

$$R = L' \qquad (4)$$
$$L = R' \oplus f(L',K)$$

Consequently, to ***decipher*** it is only necessary to apply the ***very same algorithm to an enciphered message block,*** taking care that at each iteration of the computation ***the same block of key bits*** **K** ***is used*** during decipherment as was used during the encipherment of the block.

Using the notation of the previous section, this can be expressed by the equations:

$$Rn\text{-}1 = Ln \qquad (5)$$
$$Ln\text{-}1 = Rn \oplus f(Ln,Kn)$$

where now ***R16L16*** is the permuted input block for the deciphering calculation and ***L0R0*** is the preoutput block. That is, for the decipherment calculation with ***R16L16*** as the permuted input, ***K16*** is used in the first iteration, ***K15*** in the second, and so on, with ***K1*** used in the 16th iteration.

The Cipher Function f

A sketch of the calculation of ***f(R,K)*** is given in Figure 2.2

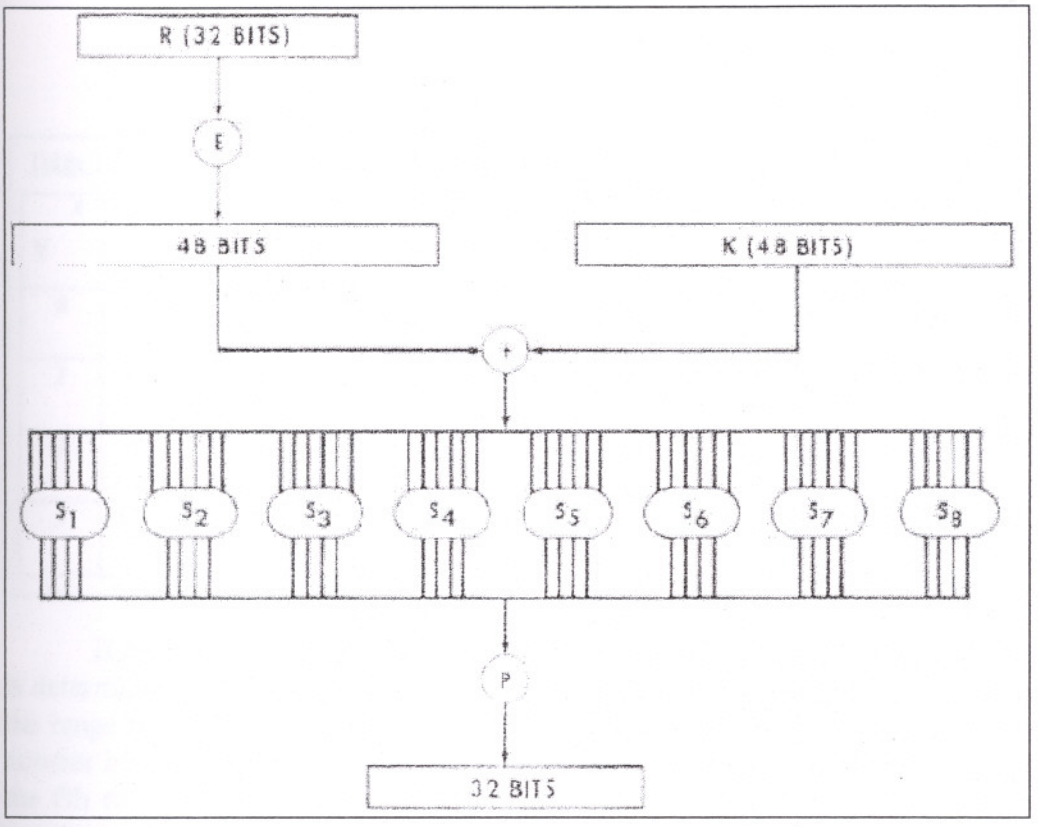

Figure 2.2 Calculation of f(R, K).

Let ***E*** denote a function which takes a block of 32 bits as input and yields a block of 48 bits as output. Let ***E*** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following Table 2.3:

Table 2.3 E Bit-Selection.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of ***E(R)*** are the bits in positions 32, 1 and 2 of ***R*** while the last 2 bits of ***E(R)*** are the bits in positions 32 and 1.

Each of the unique selection functions ***S1,S2,...,S8***, takes a 6-bit block as input and yields a 4-bit block as output and is illustrated by using the Table 2.4 containing the recommended ***S1***.

Table 2.4 S1 Function (X: Column Number Y: Row).

X / Y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If ***S1*** is the function defined in this table and ***B*** is a block of 6 bits, then ***S1(B)*** is determined as follows: The first and last bits of ***B*** represent in base 2 a number in the range 0 to 3. Let that number be *i*. The middle 4 bits of ***B*** represent in base 2 a number in the range 0 to 15. Let that number be *j*. Look up in the table the number in the *i*'th row and *j*'th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output ***S1(B)*** of ***S1*** for the input ***B***. For example, for input 011011 the row is 01, that is row 1, and the column is determined by 1101, that is column 13. In row 1 column 13 appears 5 so that the output is 0101.

The permutation function ***P*** yields a 32-bit output from a 32-bit input by permuting the bits of the input block. Such a function is defined by the following Table 2.5:

Table 2.5 Permutation Function.

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

The output ***P(L)*** for the function ***P*** defined by this table is obtained from the input ***L*** by taking the 16^{th} bit of ***L*** as the first bit of ***P(L)***, the 7^{th} bit as the second bit of ***P(L)***, and so on until the 25^{th} bit of ***L*** is taken as the 32^{nd} bit of ***P(L)***.

Now let ***S1,...,S8*** be eight distinct selection functions, let ***P*** be the permutation function and let ***E*** be the function defined above.

To define ***f(R,K)*** we first define ***B1,...,B8*** to be blocks of 6 bits each for which

$$B1B2...B8 = K \oplus E(R) \qquad (6)$$

The block ***f(R,K)*** is then defined to be

$$P(S1(B1)S2(B2)...S8(B8)) \qquad (7)$$

Thus $K \oplus E(R)$ is first divided into the 8 blocks as indicated in (6). Then each ***Bi*** is taken as an input to ***Si*** and the 8 blocks ***S1(B1),S2(B2),...,S8(B8)*** of 4 bits each

are consolidated into a single block of 32 bits which forms the input to ***P***. The output (7) is then the output of the function ***f*** for the inputs ***R*** and ***K***.

CHAPTER 3

PASSWORD STRUCTURE IN UNIX

3.1 Introduction

The reason being that UNIX systems are very popular, especially in an educational environment, where one can expect an increased concentration of hackers due to the openness that is appreciated in such environments. This is in contrast to a commercial environment, where data has to be protected. Thus enter security problems.

3.2 User Authentication

Authentication is a fancy name for identifying the user as a valid user of a given computer system, and it's the first defence against a break-in. Until recently, UNIX user authentication meant typing a valid login name and password. This is known as reusable password authentication, meaning that you enter the same password each time you log in. Reusable password authentication is too weak for some systems and will eventually be replaced by one-time password systems in which the user enters a different password each login that will need a different way of user management process.

Reusable passwords are strong enough for some sites as long as users choose good passwords. Unfortunately, many don't—as we will examine in Chapter 5 and prove in Chapter 6. A security policy should both require strong passwords and provide guidelines for choosing them.

3.2.1 Users and Passwords

The access to a Linux system is through such a user account. Every user account must be set up by the system administrator. When the Linux software is installed, one master login is created automatically, called root. This is known as the superuser because there is nothing the login can't access or do. Although most user accounts on a Linux system are set to prevent the user from accidentally destroying all the system files, for example, the root login can blow away the entire Linux operating system with one simple command. The root login has no limitations.[1]

3.2.2 Password for System Accounts

The system administrator must take special care in choosing a good password for his account and the superuser account. The superuser account must be protected because of the power it possess thus provides the cracker with, and the system administrator's account because it can give access to the superuser account in many ways. For instance, if a system administrator's account is broken, the cracker can install a fake *su* program in his private *bin* directory that records the root password,

---

[1] Tim Parker, *Linux System Administrator's Survival Guide,* Chapter 16.

removes itself, and then invokes the real *su* program.[2] The system administrator account may have other special privileges that a cracker can make use of; for instance, membership to groups that allows one to read—or worse, write—system memory or raw disk devices, and permission to su to the superuser account. The systems administrator and root passwords should be changed often and should be as strong as possible.

3.3 Password Security

3.3.1 /etc/passwd File

Every person using Linux OS should have his or her own unique user name and password stored in */etc/passwd* file and this folder contains all the information the system needs to know about each user.

If necessary levels are not set, this file can be printed out by *anyone* on the system. But having the password blocks in the /etc/passwd file is useless, the passwords are *encrypted* using an encoding scheme that makes deciphering someone's password very difficult, thanks to DES algorithm which is explained in the previous chapter. That's why system administrators prefers to give a new temporary password for a user who has lost or forgotten his or her password, instead of decrypting the old password.

A typical excerpt from /etc/passwd looks like in Figure 3.1. Each line in the /etc/passwd file is composed of seven fields separated by a colon. If nothing is to be entered in a field, the field is left blank, but the colons are retained to make sure each line has seven fields (which also means each line will have six colons).

```
johnsmith:naVwowMManasMMo:10:200:John Smith:/users/johnsmith:/bin/bash
    ^           ^         ^   ^       ^             ^           ^
    |           |         |   |       |             |           + User's
    |           |         |   |       |             |           shell program
    |           |         |   |       |             +--- User's home directory
    |           |         |   |       +---------------- User's real name
    |           |         |   +------------------------ User number
    |           |         +---------------------------- User's group number
    |           +-------------------------------------- Hash of the password
    +------------------------------------------------- Username
```

Figure 3.1 /etc/passwd File.

[2] Jeff Smith, et.al., *UNIX Unleashed,* Section 44.

- Username is the name under which the user logs in. Usually this is accomplished by typing in the username at the username prompt and then the password at the password prompt. The user name is a unique identifier for the user.

- Hash of user's password is the target of the cracking method. This is what the hash of each word in the dictionary file is compared to.

- User's group number determines things such as access to certain files, etc. Used more in the exploit technique.

- User's number is basically identification for the system. The superuser account is always defined as the account with a user ID number of zero.

- User's real name is the name the user entered. Not used by the system, but it provides a handy human-readable id of each user, sometimes it is a phone number, department, or other information.

- User's home directory is the directory that they go to when they log into the system.

- User's shell is the user interface that the user uses. Shells include /bin/bash /bin/ash /bin/tcsh /bin/csh and /bin/sh.

3.3.1.1 User Name

This is a one- to eight-character alphanumeric field that represents the user's login name. Traditionally the name is all lowercase characters. Any value may be used for this field. To make it easy to tell a user name from a userID, the name should not start with a number.

3.3.1.2 Understanding Default System User Names

The previous extract from the /etc/passwd file lists over a dozen system-dependent user names. These names serve special purposes on the Linux system. A few of these logins are worth noting as they have specific uses for the operating system and system administrators:[3]

- The root login is the superuser account (UID 0) and has unrestricted access. It owns many system files.
- The deamon login is used for system processes. This login is used only to own the processes and set their permissions properly.
- The bin login owns executables.
- The sys login owns executables.
- The adm login owns accounting and log files.
- The uucp login is used for UUCP communication access and files.
- The other system logins are used for specific purposes (postmaster for mail, and so on) that are usually self-evident. Non of the system logins should be

---

[3] Tim Parker, *Linux System Administrator's Survival Guide,* Chapter 16.

changed. In most cases, they have an asterisk in the password field to prevent their use for entry purposes.

3.3.1.3 Password (If Not Using a Shadow Password Scheme)

This field can be used to restrict access to the system. If it is wanted to prevent a login from ever being used for access, such as a system login like lp or sync, an asterisk is placed between the two colons for the password field. This asterisk restricts all access. In the /etc/passwd file, many system logins have an asterisk as their password, effectively blocking access.

This field can also be used to allow unrestricted access by leaving it blank. If no password entry exists (the field has nothing in it), anyone using the user name is granted access immediately, with no password requested.

Passwords must not be left open unless the Linux system is being used for pleasure and have nothing of value on the file system.

It must not be attempted to put a password directly in the password field using an editor. The encryption method cannot be recreated, and it will end up locking the user account out. Then only the system administrator will be able to change the password and allow access.

The users password, encrypted with a one-way cipher is stored in the second field. Only the first 8 characters of the password are used. These are mixed with a 2-character salt to produce a 13-character encrypted password.(see Section 3.3.2) When it is necessary to compare a password, the plain text is encrypted with the salt, and a comparison is made against the encrypted version. If the passwd field is empty, this account has no password, and none is required to log in.

On systems that support password aging and place the password in is the passwd file, the password data can be followed by a comma and four characters that describe the aging information. Each of these characters is drawn from the following character set:

. = 0
/ = 1
0—9 = 2—11
A—Z = 12—37
a—z = 38—63

The first character after the comma denotes the maximum number of weeks the password is valid. The second character is the minimum number of weeks required between password changes. If both of these characters are zero (..), the user is forced to change his password the next time he logs in. If the change interval is larger than the valid interval, then only the root user can change the password.

On systems that do not use the passwd file to hold the password, such as those using /etc/shadow or some password adjunct scheme, this field contains a fixed string that has fewer than 13 characters.

3.3.2 Salted Passwords

For each user's password *p*, a 'salt' value *s* is chosen at random with the real-time clock, and one-way hash function *g()* is applied to the password and the salt.[4] The value *s* and the 64-bit result of the encryption *g(p,s)* are both stored in the password file.

The salt is chosen from the set of digits, upper and lowercase letters, . (period), and / (slash). Therefore the salt is used to select one of 4,096 cryptographic methods related to the National Bureau of Standards DES encryption algorithm. That method is used to encrypt and convert the key into the 11 bytes that, along with the salt.[5]

The key search technique is still likely to turn up a few passwords when it is used on a large collection of passwords, and it seemed wise to make this task as difficult as possible.

It is likely that a bad guy (password cracker) can spend days of computer time trying to find a password on a system with hundreds of passwords, and find none at all.[6] More important is the fact that it becomes impractical to prepare an encrypted dictionary in advance that will be examined in section 4.4.3.

Salting procedure is done so that integrated circuit chips that implement DES can't be used to crack UNIX passwords.

3.3.2.1 The Threat of the DES Chip

Chips to perform the DES encryption are already commercially available and they are very fast. The use of such a chip speeds up the process of password hunting by three orders of magnitude. To avert this possibility, one of the internal tables of the DES algorithm (in particular, the so-called E-table) is changed in a way that depends on the 12-bit random number. The E-table (see Section 2.3.2) is inseparably wired into the DES chip, so that the commercial chip can not be used.[7] Obviously, the bad guy could have his own chip designed and built, but the cost would make the whole scenario unfeasible at least for the time being.

However, it would not be too expensive to build VLSI chips that compute the crypt function and run 1000 faster than these software implementations, not to mention the possible gain due to parallelism and pipelining.[8]

---

[4] Li Gong, "Protecting Poorly Chosen Secrets from Guessing Attacks", p. 3.

[5] P.H. Wood, S.G. Kochan, *UNIX System Security,* p. 36.

[6] R.Morris, K.Thompson, *Password Security: A Case History,* p. 4.

[7] R.Morris, K.Thompson, *Password Security: A Case History,* p. 4.

[8] D.C. Feldmeier, P.R. Karn, *UNIX Password Security - Ten years Later,* p. 4.

3.3.3 Login Procedure

When a user logs in, the login program logically compares the password the user typed to a block of zeros, and then compares that result to the entry in the password field. If they match, the user is granted access. Any deviation causes login procedure to refuse access.

The detailed procedure is as follows; after the login request, the 12-bit quantity is extracted from the password file and appended to the typed password. The encrypted result is required, as before, to be the same as the remaining 64 bits in the password file. This modification does not increase the task of finding any individual password, starting from scratch, but now the work of testing a given character string against a large collection of encrypted passwords has been multiplied by 4096 (2^{12}). The reason for this is that there are 4096 encrypted versions of each password and one of them has been picked more or less at random by the system.[9]

If someone wants to change his password, he can't directly modify the /etc/passwd file-that's not allowed. If it were, sooner or later someone would go in and change all the passwords; then nobody would be able to log in. Instead, he should use the passwd command.[10] All has to be done is typing in passwd and it prompts for the rest:

```
$ passwd
Changing password for pat
Old password:wizzardl                    Not printed
New password:wom2bat                     Not printed
Re-enter new password:wom2bat            Not printed
$
```

Before allowing one to change his password, the passwd command requests for typing in the old password. This is just to make sure it's really him and not some one else using your terminal while he is away. If he makes a mistake typing in the old password, the system responds with "Sorry.", meaning that no change was made and that he should try again. If the old password is correct, the passwd command then asks to enter the new password. Since the passwords are not printed, the command makes sure one doesn't unwittingly make a mistake by asking him to enter his new password a second time. If the two entries don't match, the passwd command will again ask to enter the *new* password twice:

```
$ passwd
Old password:wom2bat                     Not printed
New password:wizzardl                    Not printed
Re-enter new password: wizrdl            Not printed
They don't match; try again.
New password:wizzardl                    Not printed
Re-enter new password:wizzardl           Not printed
```

---

[9] R.Morris, K.Thompson, *Password Security: A Case History,* p. 4.

[10] P.H. Wood, S.G. Kochan, *UNIX System Security,* p. 10.

To login successfully on the UNIX system, it is necessary after dialing in to type a valid user name, and then the correct password for that user name. It is poor design to write the login command in such a way that it tells an interloper when he has typed in a invalid user name. The response to an invalid name should be identical to that for a valid name.[11]

When the slow encryption algorithm was first implemented, the encryption was done only if the user name was valid, because otherwise there was no encrypted password to compare with the supplied pass word. The result was that the response was delayed by about one-half second if the name was valid, but was immediate if invalid. The bad guy could find out whether a particular user name was valid. The routine was modified to do the encryption in either case.

3.3.4 crypt( )

crypt() is the password encrypting program on the UNIX system. It is also called by /usr/lib/makekey.[12] The crypt *routine* isn't related to the *crypt command,* crypt, like /usr/lib/makekey, takes an eight-character key and a two-character salt. The key is used as input to the setkey() routine. The salt is then used to jumble the DES algorithm in encrypt(). Finally, the encrypt() routine is called to repeatedly encrypt a constant string 25 times (to eat up computer time). crypt() returns a character pointer to the encrypted password, of which the first two characters are the salt. One of the advantages of crypt is that it uses a significant amount of computer time to encrypt a password.

Actually, crypt() does not use pure DES. To prevent use of off-the shelf high-speed DES hardware to crack passwords, crypt modifies the DES algorithm slightly. As we know the 12-bit salt ranges from zero to 4095. We can think of the salt as a permutation that immediately follows the expansion function *E* in DES. If bit 1 of the salt is a 1, then the salt permutation swaps bits 1 and 25 of the 48-bit block generated by *E*. If bit 2 is a 1, then bit 2 and 26 are swapped and so on.[13] Since there are 12 possible swaps and any combination of these swaps may occur, this produces 4096 possible variations of DES (a salt of zero corresponds to pure DES).

3.3.5 Shadow File Entry

Since the /etc/passwd file is world readable, as an added measure of security, SVR4 UNIX systems, (also used in current versions of Linux systems available) use a shadow file to hold the password information.[14]

It is readable only by root because of potential security problems against crackers. It contains the password field data in an expanded format and the password fields on the system are all set to x.

---

[11] R.Morris, K.Thompson, *Password Security: A Case History,* p. 4.

[12] P.H. Wood, S.G. Kochan, *UNIX System Security,* p. 81.

[13] D.C. Feldmeier, P.R. Karn, *UNIX Password Security - Ten years Later,* p. 11.

[14] Tim Parker, *Linux System Administrator's Survival Guide,* Chapter 6.

The shadow file, as shown in Table 3.1, is not designed to be edited directly, but instead is modified by the passwd command automatically as needed.

Table 3.1 Excerpts from a sample /etc/shadow file from an SVR4 system.

```
root:03de466J423f5:6445::::::
daemon:NP:6445::::::
bin:NP:6445::::::
sys:NP:6445::::::
adm:NP:6445::::::
lp:NP:6445::::::
smtp:NP:6445::::::
uucp:NP:6445::::::
nuucp:NP:6445::::::
listen:*LK*:::::::
Pwcsite:x3d5dtyfetonK:8774::::::
syd:43ASxete436h.:8776:0:168:7:::
nobody:NP:6445::::::
noaccess:NP:6445::::::
```

The shadow file consists of the following fields:

User Name

This name is used to match against the name in the passwd file.

Password

The user's password, encrypted with a one-way cipher, is stored in the second field. This field has the same properties as in the /etc/passwd file Only the first 8 characters of the password are used. These are mixed with a 2-character salt to produce a 13-character encrypted password. When it is necessary to compare a password, the plain text is encrypted with the salt, and a comparison is made against the encrypted version. If the passwd field is empty, the account has no password, and none is required to log in.

CHAPTER 4

VARIOUS ATTACK SCHEMES ON PASSWORD

4.1 Introduction

The password is the most vital part of account security. If an attacker can discover a user's password, he or she can then log into the system and operate with all the capabilities of that user. Such an attack is usually hard to detect and can last for months.

Easy-to-guess passwords offer hackers the possibility to enter a system. By means of good password-security, one can protect a system from newbie hackers.

There are many ways to hack a UNIX system, and there are many programs for finding a user's password. These programs can be used by people who have little knowledge of UNIX. Choosing good passwords can therefore help in keeping newbie hackers out. Advanced hackers are often capable of entering a system without using passwords.[1] This implies that the security of a system depends not solely on well-chosen passwords.

In UNIX one can use all printable characters, case is significant and only the first 8 characters will be used, so in the password `computer+3,Z' only `computer' (an easy-to-guess password) is significant and the remaining characters will be ignored. On some systems like Ultrix with upgrade security features one can have passwords with up to 16 characters.

4.2 The Importance of Good Passwords

The security of each individual user is closely related to the security of the whole system. Users often have no idea how a multi-user system works and don't realize that they, by choosing an easy to remember password, indirectly make it possible for an outsider to manipulate the entire system. It is essential to educate the users well to avoid attitudes such as below.[2]

"It doesn't matter what password I use on my account, after all, I only use it for laser printing..."

It is important to notify the users of the security guidelines. A solution might be giving new users a limited course. Or at least make them understand why good

---

[1] Walter Belgers, *UNIX Password Security,* p. 1.

[2] Alec E. Muffett, *Almost Everything You Wanted To Know About Security,* Internet Document, USENET newsgroup.alt.security.

passwords are essential. This can be done e.g. when a user gets his or her initial password from the system administrator.[3]

We see that the usual way to find passwords is by guessing them. So we have to make sure that users do not use easy-to-guess passwords, i.e. passwords that can be found in lists (a dictionary, an encyclopedia, files with astronomical terms, flora and fauna, etc.).

4.3 Human Password Choices and System Security

Encryption is a vital security component. However, no matter how strong the encryption is, it will fail when users make poor password choices. Users are lazy, error-prone, and forgetful.[4] Usually, users create passwords from the following (partly to save time and make their lives easier):

Birth date

Social security number

Children's names

Names of favorite performing artists

Words from the dictionary

Numeric sequences (like 90125)

Words spelled backwards

By regularly checking the strength of the passwords on the network, one can ensure that crackers cannot penetrate it by exploiting bad password choices. Such a regimen can greatly improve the system security. In fact, many system administrators now employ tools that check a user's password when it is first created.

4.4 Password Attacks

Password security is so critical that the system will never be safe without it. Indeed, one could install a dozen firewalls and still, if her/his passwords were vulnerable, Linux system would have an open door.

Hence, password security demands practically a two-pronged approach. On the one hand, we can apply advanced tools to strengthen passwords. On the other, we can educate our users and hold them responsible to essential password policies.[5]

In the security pecking order, password attacks are primitive. In fact, password cracking is the first thing that budding hackers and crackers learn, chiefly

---

[3] Walter Belgers, *UNIX Password Security,* p. 1.

[4] Anonymous, *Maximum Linux Security,* pp. 154-155.

[5] Anonymous, *Maximum Linux Security,* p. 122.

because it demands minimal technical expertise. Today, anyone can crack Linux passwords using automated tools.

Attackers that initially gain only limited access can rapidly expand that access by attacking weak password security. Often, through password attacks alone, attackers obtain root access and seize control of not just one host but several.

This chapter will cover various password dictionary attack techniques, importance of passwords, predefined cases for such a dictionary attack, and steps required to secure the passwords, including advices.

4.4.1 Dictionary Attacks

DES, like most things, is not infallible. Linux passwords encrypted with DES can be cracked quickly, usually within minutes. There are two chief reasons for this:[6]

The human factor: Users invariably choose characteristically weak passwords.

Limited length: Linux passwords are short. The number of transformations necessary to encrypt one is relatively small.

In dictionary attacks, attackers take dictionaries or long wordlists and encrypt them using DES. During this process, they send regular words, proper names, and oilier text through precisely the same permutations and transformations that Linux passwords are exposed to. Over time, using high-speed cracking tools, attackers can encrypt each dictionary word in some 4,096 different ways. Each time a cracking tool derives such encrypted text, it compares it to the passwords from /etc/passwd. Sooner or later (often sooner) it finds a match, and when it does, it notifies the attacker; that a password has been cracked.

4.4.2 Fast Crypt Implementations

The crypt implementation that is included with UNIX distributions (such as BSD 4.2) is not optimized for speed because it already allows logins in a reasonable amount of time. Several techniques can be used to speed up an implementation. One technique is to alter the crypt algorithm so that it is easier to compute but still produces the same results. Another technique is to take advantage of the architectural features of the computer that runs the algorithm.[7] Some of the cracker programs use these features.

Using the speeds of several fast crypt implementations and the prices of several computers an effective ratio can be found. Crypts/seconds/dollars is the correct metric because password cracking is an easily segmented problem.

The ultimate size of the key space allowed by the UNIX crypt program is very large: 2^{56} or about 7.2×10^{16} possible keys. Even with only 95 printable characters on

---

[6] Anonymous, *Maximum Linux Security,* p. 128.

[7] D.C. Feldmeier, Philip R. Karn, *UNIX Password Security - Ten Years Later,* p. 2.

a keyboard, there are still 95^8 or about 6.6×10^{15} possible keys. This is large enough to resist brute-force attacks in software, yet most of the passwords selected by users are in a very small part of this total space.

4.4.3 Precomputed Encrypted Dictionaries

A fast way of cracking large batches of passwords on a routine basis is to first encrypt a list of likely passwords and then compare each new batch of encrypted passwords against this pre-encrypted list. Salting was specifically designed to hinder this approach. Because the specific salt values are not known in advance, the pre-encrypted dictionary must encrypt each trial password with all possible salts, increasing storage requirements considerably.

Encrypting each trial password 4,096 times (once for each possible salt value) takes several CPU-weeks. Each encrypted password is stored as an 8-byte value; the plain text is not stored on the tape. Not only does this reduce the amount of tape necessary, but the tapes alone are enough to determine whether an encrypted password is in the password list without revealing the plain text password. This is ideal for improving system security without the possibility of the tapes being used to infiltrate other systems.[8]

The cassettes can be replayed repeatedly and checked against lines from the /etc/passwd file. The system checks faster than the fast crypt code runs in real time. The tapes also can be supplemented with tapes produced from new passwords. The precomputed dictionary is helpful but not essential for password cracking.

4.5 Automated Dictionary Attack

The elements required to crack passwords using any of the corresponding crackers are:[9]

- High performance/price ratio computers
- Large on-line word lists (dictionaries, etc.)
- A known password encryption algorithm (DES)
- A constraint on the acceptable running times for the login program
- A publicly-readable password file
- Passwords with a significant probability of being in the word list

---

[8] D.C. Feldmeier, Philip R. Karn, *UNIX Password Security - Ten Years Later,* p. 4.

[9] D.C. Feldmeier, Philip R. Karn, *UNIX Password Security - Ten Years Later,* p. 5.

4.5.1 Known Encryption Algorithm

We consider it a given that the encryption algorithm used for the one-way password crypt function must be published and subjected to public scrutiny. As in cryptography, it is neither practical nor necessary to base the security of a password algorithm on its secrecy. The storm of protest in response to the NSA's recent attempt to replace DES with a secret cipher of its own design indicates the importance of this principle. Furthermore, the enormous success of the UNIX operating system is based largely on the openness of its design and the availability of its algorithms and source code. Assuming that the basic algorithm has not been compromised, there is no real reason to change it.[10]

4.5.2 Acceptable Running Times

Software de-facto standards, such as the UNIX password algorithm, tend to outlive their original underlying hardware. Also, a crypt routine written specifically for password cracking runs orders of magnitude faster than a version built into a login command.

4.5.3 Encrypted Password Availability

A resource available to the adversary that is removable is the existence of a, publicly readable encrypted password file (/etc/passwd). It is assumed that physical access to the machine alone is enough to subvert it and it is assumed that the machine itself is physically secured according to the desired level of security. Many machines can be rebooted into privileged mode with physical access, so that physical access implies that a password-based attack is really unnecessary for system access.

4.5.4 Decreasing Password Guessability

The main weakness in any password system is that users often choose easily guessable passwords. One way to decrease password guessability is to eliminate common passwords from /etc/passwd.

Another possibility is to restrict the passwords accepted from the user with a system that filters out easily guessed passwords. This system acts as a password advisor that indicates insecure passwords, but it does not force the user to accept its recommendation.

The most drastic solution is to have the system assign an arbitrary password. The problem is that such a password is hard to remember, so the temptation to write it down is strong. A written password is like a physical key, and can be used by anyone who obtains it.

A fundamental problem is that passwords typed by the user are truncated to 8 characters in length. Easily remembered passwords that are this short almost inevitably have much less than the 56 bits of entropy allowed by the crypt algorithm, making them easier to find by exhaustive search. All of the techniques just described

---

[10] D.C. Feldmeier, Philip R. Karn, *UNIX Password Security - Ten Years Later,* pp. 5-8.

attempt to increase entropy in the users' passwords, but they do it in a way that ignores human factors considerations. Almost anyone can remember 56 bits of arbitrary information, but he must, be allowed to do it in a way that is suited to human, not computer, memory. The way to do this is by extending the present algorithm to allow *pass phrases.*[11] A pass phrase is simply a longer version of a password that includes several words. According to Shannon[12], English text has a lower bound of 1-2 bits of entropy per character. Therefore an ordinary English phrase of 5-10 words (assuming 5-6 characters/word and no unusual punctuation or capitalization) has sufficient entropy as a pass phrase.

To accommodate this in the UNIX crypt algorithm, a hash function is needed to fold the typed pass phrase into 56 bits, with each input character affecting the result. This function should be backward compatible with the existing UNIX password algorithm for pass phrases of 8 characters or less. One possibility is to treat the first 8 characters as before, exclusive-ORing into it each successive 8-character block from the pass phrase (if the phrase is not a multiple of 8 characters, it is null-padded on the right).

Users might still object to pass phrases if they were required to type them too frequently (e.g., when they must repeatedly log into a several different systems, each for short intervals). A solution to this problem lies in the use of an distributed authentication system such as Kerberos, in which the user needs to type his password only once to obtain a set of "tickets" that can be used to access other systems repeatedly without having to retype the pass phrase each time.[13]

4.5.5 Other Approaches

Two suggested solutions to the problem of easily cracked passwords are to increase the size of the salt or to change the constant that is encrypted by crypt. Neither of these seems to be particularly helpful.

Increasing the size of the salt does not help prevent attack on an individual password, but it does help defeat checking multiple passwords simultaneously and pre-encrypted wordlist attacks by increasing the time and space required, respectively. The current salt is large enough that few of the lines in a typical /etc/passwd file share the same salt. The only remaining reason to increase the size of the salt is to reduce the number of pre-encrypted passwords that can fit onto a fixed amount of tape. But as shown, pre-encryption decreases the cracking time by a factor of 30, so this is the maximum penalty that could be exacted by even a large increase in the salt size.

The current UNIX password system is not always sufficient to prevent unauthorized entry because it is fairly easy to crack passwords. An important point is that although the crypt algorithm is a good one, the password system as a whole is

---

[11] D.C. Feldmeier, Philip R. Karn, *UNIX Password Security - Ten Years Later,* p. 7.

[12] Claude Shannon, *"Prediction and Entropy of Printed English. Bell System Technical Journal".*

[13] J.G. Steiner, et.al., "Kerbelos, *An Authentication Service for Open Network Systems".*

weak. Nothing can be done about large on-line dictionaries or high performance/price ratio computers. The password encryption algorithm must be known to be trusted and that there is a range of acceptable running times for the algorithm which sets an upper limit on the amount of computation that the password encryption algorithm may use. Unfortunately, the computation limit is small enough to allow faster machines to use a dictionary-based attack.

Two of the main problems with the current system are that users choose easily guessable passwords and that the encrypted password file is publicly readable. A dual approach is suggested. One part is to make passwords less predictable by allowing pass phrases and restricting passwords accepted by the system. This effectively increases the entropy of a password, making wordlist attacks less successful. The other approach is to make the encrypted password file less accessible. How exactly this is done depends on the desired level of security and includes shadow password files.

4.6 Password Composition Vulnerabilities

There are really only a couple of problems with passwords: Picking a good one, and then managing it. "Good" passwords avoid being something that an intruder can guess or otherwise easily figure out.

4.6.1 Bad Passwords

Bad passwords have the following properties:

Exactly match a word in the dictionary,

Match a reversed word in a dictionary,

Match a word in the dictionary with some or all of the letters capitalized,

Match a reversed word in the dictionary with some or all of the letters reversed.,

Are shorter than a specific length (usually 6 characters),

Do not contain a mix of upper and lower case, or mixed letters and numbers, or mixed letters and punctuation,

Are based on the users account name, initials, or given name, or any other info about the user: SSN, license plate number, etc.,

Match a dictionary word with any of the following translations:

a -> 2, a -> 4, e -> 3, h -> 4, i -> 1, l -> 1, o -> 0, s -> $, s -> 5, z -> 5,

Are conjugations or plurals of dictionary words,

Are acronyms, geographical or product names, and technical terms,

Are either proceeded or followed by a digit, a punctuation mark, up arrow, or space,

Are a word with all the vowels deleted,

Are phrases with the whitespace deleted,

Are all numbers.

4.6.1.1 What Not to Use as a Password

Words similiar the login name in any form (as-is, reversed, capitalized, doubled, etc.).

Words that are also first or last name of the user in any form.

Spouse's or child's name.

All digits, or all the same letter. This significantly decreases the search time for a cracker.

Word contained in (English or foreign language) dictionaries, spelling lists, or other lists of words.

One of the above with a single character tacked onto the end.

Password shorter than six characters.[14]

Words that can be found in the password file itself like Okyar, Berkay, Adam, etc.

Patterns like 123456, qwerty, etc.

Geographical names

Words from an encyclopedia ('Socrates')

The license plate of a car, the room number, the phone number or other things that have something to do with the owner of the account

Given names

Variations such as walter, WALTER, retlaw, Walter, wAlter, walter0, walt3r, Retlaw4,...

Acronyms of words that are in any dictionary of any language, spelling lists, or other lists of words.

Sequences of letters like 'abcdef' or 'qwerty', place names, car names, cartoon heroes.

Only the first or the last character in uppercase [15]

Only vowels in uppercase

Only consonants in uppercase.

---

[14] David A. Curry, *Improving The Security of Your UNIX System*, p. 7.

[15] Lionel Cons, *CERN Security Handbook Practical Computer Security for CERN Users Version 1.2*

A character appended or prepended to a word from a dictionary (for instance `7tables' or `secret!') or use simple substitutions like o==>0 or s==>$ (for instance `sn00py' or even `$n00py'), most cracking programs will also try these...

Some characters are "dangerous" for passwords because they can be trapped by some programs, one should therefore not use them in his passwords. They include:[16]

Most "control" characters such as Ctrl-C or Alt-Q and other like:

- #: can be interpreted as **erase** by some versions of telnet
- @: can be interpreted as **kill** by some versions of telnet
- \: is usually an "escape" character, for instance `\#' may be interpreted as a simple `#' (and not the **erase** character).

4.6.2 Good Passwords

Might be two words separated by a non-letter non-digit, such as 'mac2%beav' or 'cat,bear#'. Note that 'go2work' is probably bad. A good password is:[17]

private: it is used and known by one person only

secret: it does not appear in clear text in any file or program or on a piece of paper pinned to the terminal

easily remembered: so there is no need to write it down

not guessable: by any program in a reasonable time, for instance less than one week.

Although this seems quite restrictive, it's easy to pick good passwords.

4.6.2.1 What to Use

Use a password with:

- mixed upper- and lower-case alphabetic.

- non-alphabetic characters, e.g., digits or punctuation.

Use a password that:

- is easy to remember, so there will be no need to write it down.

---

[16] Lionel Cons, *CERN Security Handbook Practical Computer Security for CERN Users Version 1.2*

[17] Lionel Cons, *CERN Security Handbook Practical Computer Security for CERN Users Version 1.2*

- can be typed quickly, without having to look at the keyboard. This makes it harder for someone to steal the password by watching over his shoulder.

4.6.2.2 Reducing Break-in Possibilities

It is important for users to have hard to guess but in the meantime easy to remember passwords. There are methods for generating such passwords. System operators should inform the users about the importance of good passwords.[18] To reduce the risk of a break-in there are several possibilities:

- Making sure the users know why a good password is important and how they can choose one.

- Installing a new /bin/passwd (or yppasswd) that checks whether the password is not too obvious (by checking if it contains punctuation marks, or by investigating if the password can be found in standard wordlists).

- Installing a shadow password file (this involves changing some software).

- Letting passwords expire, for example after three months for regular users, after a month for users with extra privileges. The timespan a password lives should not be chosen too small. What will still exist is the danger of users using series of passwords, like 'Secret1', 'Secret2',... making it easy for a hacker to, once he has obtained a password, guess the successor.

- Using a program that hacks passwords to check if some users have guessable passwords. Letting those users visit the administrator personally to inform them about the fact that a good password is everyone's concern.

- Switching to single-use passwords.[19]

- Using passwords of accounts with privileges like that of root on the console only to avoid eavesdropping the network. When impossible, trying to avoid logging in on such accounts from computers or terminals that are connected to a LAN segment, on which people can easily and/or anonymously wiretap the network, like classrooms.

It must be kept in mind that *the total system security is as weak as the weakest chain.*

4.6.2.3 Password Screening

Retroactive password vetting puts the administrator in the role of the cracker. The admin makes the best effort to break the users' passwords, and if he succeeds he notifies the user and require her to change her password to something safer.[20]

---

[18] Walter Belgers, *UNIX Password Security.*

[19] Wietse Venema, *Using SecurID Tokens in an Open Multi-Host UNIX Environment,* Internet Document, ftp.nic.surfnet.nl as /surf/net/net-security/docs/securid, 1993.

[20] Jeff Smith, et.al., *UNIX Unleashed,* Section 44.

Proactive password screening is more like a preemptive strike—the users are prevented from choosing poor passwords. With proper education via a security policy users will react more positively to being told they must choose a more secure password than to being told that their current one is broken.

4.6.2.4 Picking Good Passwords

The object when choosing a password is to make it as difficult as possible for a cracker to make educated guesses about what one's chosen. This leaves him no alternative but a brute-force search.

Good passwords are 6—8 characters long, use a rich character set (upper and lowercase letters, digits, punctuation, and control characters), are not in Turkish or any foreign-language dictionaries, and don't contain any public information about the user, such as his name or license number. One good method is to take a random phrase and modify it in ingenious ways. For instance, the phrase "If pigs had wings" could yield the password "1fpiGzhw." This password is a combination of a misspelled word ("1f" standing for "if"), a misspelled word with odd capitalization ("pigZ"), and the first letters of two more words. It's as secure as a reusable password can be since it isn't found in any dictionary and uses a fairly rich vocabulary (the digit "1" and capitalization in a system which lets the first character of the password to be a digit), and it's easy to remember.[21]

Password choice is one of the areas in which users will deviously (and sometimes maliciously) thwart the security policies—some people can't be convinced that they should pick a good password. There are two alternatives for these recalcitrant users: proactive and retroactive password vetting as explained above.

4.6.2.4.1 Method to Choose Secure and Easy to Remember Passwords

Concatenate two words that together consist of seven characters and that have no connection to each other. Concatenate them with a punctuation mark in the middle and convert some characters to uppercase, for instance: 'Pit+idEa', 'plOVer#me'.[22]

Use the first characters of the words of a certain (not too common) sentence.

Alternately pick a consonant and one or two vowels resulting in a pronounceable (and therefore easy to remember) word. Examples: 'koDupaNy', 'eityPOop'.

Use a password with mixed-case alphabetic, digits, and punctuation.

Use long passwords (with more than 6 characters).

Finally, here are some methods of making passwords:[23]

---

[21] Jeff Smith, et.al., *UNIX Unleashed,* Section 44.

[22] Walter Belgers, *"UNIX Password Security".*

[23] Lionel Cons, *CERN Security Handbook Practical Computer Security for CERN Users Version 1.2*

Choose a line or two from a song or poem, and use the first letter of each word. For example, `In Xanadu did Kubla Kahn a stately pleasure dome decree' becomes `IXdKKaspdd'.

Alternate between one consonant and one or two vowels, up to eight characters, do use mixed-case. This provides nonsense words that are usually pronounceable and thus easily remembered (ex: `roUtboo', `quADpop', and so on.).

Choose two short words (or a big one that can be splited) and concatenate them together with one or more punctuation characters between them (or digits if only alphanumeric characters can be used). For example: `dog+F18' or `comP77Uter'. Note that `dog', `F18' or `computer' are in dictionaries but as the passwords use punctuation or digit, mixed-case characters, they are really hard to guess.

4.6.2.5 Improving Passwords Against Social Engineering

When system users have such passwords, even the best security systems cannot protect against intrusion. What makes a strong password (one that is difficult to break)? Here are a few general guidelines that many system administrators adhere to:[24]

It must be avoided:

- using any part of a user's real name and any name from the user's family or pets (these passwords are the easiest to guess).

- using important dates (birth dates, wedding day, and so on) in any variation.

- numbers or combinations of numbers and letters with special meaning (license plate number, telephone number, special dates, and so on).

- any place names or items that may be readily identified with a user (television characters, hobby, and so on)

Producing a strong password isn't that difficult. The users must be forced into the habit of mixing letters, numbers, and characters at random. Suppose a user wants to use lionking as a password. Encouraging modification to lion!king!, l_ionk_ing, lion5king, or some similar variation can be needed. Even a slight variation in a password's normal pattern can make life very difficult for someone trying to guess the password.

The /etc/passwd file can be checked at regular intervals to see whether there are entries that may have been added as a route into the system without being recognized. It must also be checked whether each account has a password. Any accounts that are not needed anymore should be removed.

---

[24] Tim Parker, *Linux System Administrator's Survival Guide,* chapter 24.

4.7 Case Studies: Cracking Linux Passwords via Dictionary Attack

A dictionary attack experiment made on a password file gives disappointing results. Some words produced were from the set of :[25]

- The dictionary with the words spelled backwards.

- A list of first names (best obtained from some mailing list). Last names, street names, and city names also work well.

- The above with initial upper-case letters.

- All valid license plate numbers.

- Room numbers, social security numbers, telephone numbers, and the like.

4.7.1 L0phtCrack 2.5

L0phtCrack is an NT password auditing tool. It will compute NT user passwords from the *cryptographic hashes* that are stored by the NT operating system. The operating system does not store the user passwords in their original clear-text form for security reasons. The actual user passwords are encrypted into hashes because they are sensitive information that can be used to impersonate any user, including the administrator of the operating system. L0phtCrack computes the password from a variety of sources using a variety of methods.[26]

There are many uses for computing user passwords. First and for most it is for a system administrator to check the strength of the passwords that their users are using. Other uses include recovering a forgotten password, retrieving the password of a user in order to impersonate them, or migrating NT users to another platform such as UNIX.

L0phtCrack 2.5 is distributed in a self-installing executable distribution file. When the installation file is run it will create a directory named \Program Files\L0phtCrack, put itself in and add a L0phtCrack start menu item.

L0phtCrack can recover passwords directly from the registry, from the file system and backup tapes, from repair disks, or by recovering the passwords as they traverse the network. L0phtCrack first extracts the password hashes, which is the way the OS stores the encrypted passwords. It uses three different methods for cracking.

The fastest method for cracking the passwords is a *dictionary attack.* L0phtCrack tests all the words in a dictionary or word file against the password hashes. When it finds the correct password it displays the result. L0phtCrack ships with a small but effective word file.

---

[25] R.Morris, K.Thompson, *Password Security: A Case History,* p. 3.

[26] *L0phtCrack 2.5 Manual.*

The second method L0phtCrack uses is called a *hybrid* crack method. This builds upon the dictionary method by adding numeric and symbol characters to dictionary words. Many users choose passwords such as "bogus11" or "Annaliza!!". These passwords are just dictionary words slightly modified with additional numbers and symbols. The hybrid crack rapidly computes these passwords. These are the types of passwords that will pass through many password filters and policies yet still are easily crackable.

The final and most powerful cracking method is the *brute force* method. This method will always recover the password no matter how complex. It is just a matter of time. Really complex passwords that use characters that are not directly available on the keyboard may take so much time that is not feasible to crack them on a single machine using today's hardware. But most complex passwords can be cracked in a matter of days. This is usually much shorter than the time most administrators set their password policy expiration time to. Using a real-world cracking tool is the only good way to know what time one should set for password expirations.

Even though getting the password files for the corresponding OS's is not a focus of this thesis; how to get a copy of the hash file is explained here within the L0phtCrack. In advance this will also be a good example for the ones who think retrieving a copy of hashes is far away from practice.

L0phtCrack must first retrieve the password hashes to start the cracking process. If one has administrator rights he can use the Tools Dump Passwords from Registry command on the L0phtCrack menu to retrieve the hashes. One can dump the password hashes from the local machine or over the network if the remote machine allows network registry access. The NT machine name or IP address is entered into the Dump Passwords from Registry dialog box and OK is pressed. The usernames and password hashes are now loaded into L0phtCrack. If this is the way one has retrieved the password hashes he may now proceed to crack the password hashes.

The second method is to access the password hashes from the file system. Since the operating system holds a lock on the Security Accounts Manager (SAM) file where the password hashes are stored on the file system it is not possible to just read them from this file while the operation system is running. Sometimes a backup of this file is made on tape or on an Emergency Repair Disk or in the repair directory of the system hard drive. Also, another operating system such as DOS can be booted from a floppy and the password hashes can be read directly from the file system. This is especially useful if physical access to the machine is possible and it has a floppy drive.

Password hashes can be loaded from a "SAM" or "SAM._" file into L0phtCrack by using the File Import SAM File menu command and specifying the filename. L0phtCrack will automatically expand compressed "SAM._" files on NT.

The final method L0phtCrack offers is to capture the encrypted hashes over the network. Ones machine must have one or more Ethernet devices to access the network. Server Message Block (SMB) Packet Capture command is used to bring up the SMB Packet Capture window. User will now be capturing any SMB

authentication sessions that the network device can capture. If the computer is on a switched network it will only see sessions originating from the machine or connecting to the machine.

As SMB session authentications are captured they are displayed in the SMB Packet Capture window. The display shows source and destination IP addresses, the user name, the SMB challenge, the encrypted LANMAN hash and the encrypted NTLM hash, if any. The capture can be saved at any time using the Save Capture button. To crack these hashes one must save the capture and then open the captured file using the File Open Password command. Other passwords be captured and cracked at the same time.

The first method L0phtCrack uses to crack passwords is called a *dictionary attack*. This method tries to encrypt each word in a dictionary or word file. It then tests each encrypted word against the password hash. If it gets a match it knows the user's password is that dictionary word. L0phtCrack comes with a nice 25,000-word file named *words-english* that contains many common words. This file or another word file is loaded into L0phtCrack using the File Open Wordlist File menu command. The default dictionary file is the words-english file.

Next select Tools Run Crack on the menu to start the cracking process. The default options for cracking are to run a *dictionary attack*, then a *hybrid attack*, and then the *brute force attack*. L0phtCrack runs these attacks on the password hashes in succession by default.

During any crack attack the L0phtCrack window displays status information to show the progress of the attack. During dictionary attacks the number of dictionary words tried is displayed along with the percentage complete.

After the dictionary attack is completed the *hybrid attack* begins. The hybrid attack uses simple patterns that users use when creating passwords from common words. By slightly modifying dictionary words the way users do, L0phtCrack is able to make educated guesses to decide which passwords to try. An example would be to try 'BOGUS11'. Many users just append a few numbers or symbols to a dictionary word in an attempt to make it a non-guessable password. L0phtCrack can guess these passwords quickly. In much less time than it would take for a brute force attack. L0phtCrack 2.5 checks to see if any number of number and symbol characters are appended to each word in the word file that has been selected. The default number of number and symbol characters is 2. This can be changed using the Tools Options command.

After the dictionary and hybrid attacks have completed the *brute force attack* begins. Brute force can take a long time but it usually takes far less time than most password policies specify for password changing. This makes passwords found during the brute force attack still too weak. The character set can be configured that the brute force attack uses with the Tools Options command. The default character set is all the alphanumeric characters and the numbers 0 through 9.

One can expect the brute force attack to take of 24-72 hours on machines with CPUs ranging from Pentium II/450 to Pentium 166. L0phtCrack has useful commands to help the user through the cracking process. It is easy to open files, run and configure profiles for the job.

4.7.2 John the Ripper

John the Ripper is a password cracker, currently available for UNIX, DOS, WinNT/Win95. Its primary purpose is to detect weak UNIX passwords. It has been tested with Linux x86/Alpha/SPARC, FreeBSD x86, OpenBSD x86, Solaris 2.x SPARC and x86, Digital UNIX, AIX, HP-UX, and IRIX. The DOS and Win32 ports are done with DJGPP and Cygnus Developer's Kit, respectively.[27]

To run John, it must be supplied with some password files and optionally specify a cracking mode, like this, using the default order of modes, and assuming that passwd is a copy of the password file:

```
john passwd
```

or, to make it use a wordlist with rules only:

```
john -wordfile:/usr/dict/words -rules passwd
```

Cracked passwords will be printed to the terminal and saved in file called ~/john.pot. This file is also used not to load passwords that one has already cracked, when he runs John the next time. To retrieve the cracked passwords, run:

```
john -show passwd
```

While cracking, any key can be pressed for status, or Ctrl+C to abort the session, saving point information to a file (~/restore by default). By the way, Ctrl+C is pressed twice John will abort immediately without saving. The point information is also saved every 10 minutes (configurable in the configuration file, ~/john.ini) in case of a crash. To continue an interrupted session, run:

```
john -restore
```

John the Ripper is designed to be both powerful and fast. It combines several cracking modes in one program, and is fully configurable for particular needs (one can even define a custom cracking mode using the built-in compiler supporting a subset of C). Also, John is available for several different platforms, which enables the same cracker to be used everywhere (for example even continue a cracking session that was started on another platform).

John supports (and autodetects) the following ciphertext formats: standard and double-length DES-based, BSDI's extended DES-based, FreeBSD's (and not only) MD5-based, and OpenBSD's Blowfish-based. With just one extra command to extract the passwords, John can crack AFS passwords and WinNT LM hashes.

---

[27] *John the Ripper-Password Cracker Manual.*

Unlike other crackers, John doesn't use a crypt(3)[28]-style routine. Instead, it has its own highly optimized modules for different ciphertext formats and architectures. Some of the algorithms used couldn't be implemented in a crypt(3)-style routine: they require a more powerful interface (bitslice DES is an example). Additionally, there're assembly routines for several processors and architectures (special Intel Pentium version, x86 with MMX, generic x86, Alpha EV4, SPARC V8).

4.7.3 Crack

Crack is the UNIX community's best-known password auditing tool that is designed to quickly locate insecurities in UNIX (or other) password files by scanning the contents of a password file, looking for users who have misguidedly chosen a weak login password.[29]

In early releases, its author. Alec Muffett, described Crack as;

> "...a freely available program designed to find standard UNIX eight-character DES encrypted passwords by standard guessing techniques... It is written to be flexible, configurable and fast, and to be able to make use of several networked hosts via the Berkeley rsh program (or similar), where possible."

Over time, he only slightly amended that description. Today, Muffett describes Crack as

> "...a password guessing program that is designed to quickly locate insecurities in UNIX (or other) password files by scanning the contents of a password file, looking for users who have misguidedly chosen a weak login password."

Some new features of Crack are:

- Complete restructuring - uses less memory
- Ships with Eric Young's "libdes" as standard
- API for ease of integration with arbitrary crypt() functions
- API for ease of integration with arbitrary passwd file format
- Considerably better gecos-field checking
- More powerful rule sets
- Ability to read dictionaries generated by external commands
- Better recovery mechanisms for jobs interrupted by crashes
- Easier to control (e.g.: to put to sleep during working hours)
- Tested on Solaris, Linux, FreeBSD, NetBSD, OSF and Ultrix [30]

---

[28] Simson Garfinkel, *Practical UNIX Security,* p 30.

[29] Alec Muffett, *Crack Version v5.0 User Manual.*

[30] Alec Muffett, *Crack Version v5.0 User Manual.*

And the requirements of using Crack are as follows:

- UNIX-like operating system.
- C Compiler
- Moderate amount of disk space.
- Lots of CPU time.
- Permission from the sysadmin.
- Root-privileges, quite possibly.
- "gzip" is extremely desirable.
- "perl", if networking/multiprocessing.[31]

Unpack the Crack distribution.

Edit the "Crack" script, configuring the values of CRACK_PATH, C5FLAGS, CC, CFLAGS and LIBS to suit the operating system.

The general form to invoke Crack is:

Crack [options] [-fmt format] [file ...]
e.g.: Crack -nice 10 /etc/passwd

...where "filename" is a file that stores password entries, e.g.: "/etc/passwd".

Crack does not generate human-readable output directly; instead, to see the results of a Crack run, the user should do:

./Reporter [-quiet] [-html]

The "-quiet" option suppresses the reporting of errors in the password file (corrupt entries, etc), whilst "-html" produces output in a fairly basic HTML-readable format.

4.7.4 Other Linux-Compatible Password Auditing Tools

Killer Cracker

A lightweight password auditing tool from Doctor Dissector, written in C++. Although Killer Cracker lacks some of the extended functionality available with Crack, it's still fast.

Lard

A password auditing tool for Linux and other UNIX versions. Lard is small enough to fit on a floppy diskette, which is good for auditing on non-networked boxes in different departments.

---

[31] Alec Muffett, *Crack Version v5.0 User Manual.*

PerlCrack

A Perl DES password cracker for Linux.

Xcrack

A Perl script for cracking Linux passwords. It does not exercise complex rules. Instead, it performs straight-ahead encryption of words for the wordlist.

Some of the password auditing tools use brute force as well. A brute force attack will always eventually prevail("Eventually" here could mean months). Conversely, a dictionary attack is only as good as the wordlist and the rules that are defined to use the words in the wordlist.[32]

[32] Anonymous, *Maximum Linux Security,* p. 136.

CHAPTER 5

TURKEY TURKISH BASED PASSWORD CRACKER

5.1 Design of the Utility

In the application part of this thesis, a Turkish password cracker is designed and coded in C language on a Linux operating system. This application aims to prove the weak password choices of human that are described and classified in section 4.3. and to find its percentage in the whole account table of a real world UNIX system. During the design of the cracker, the most suitable software and hardware environments are investigated. C programming language is selected for the coding medium and Linux distribution SUSE 6.2 is selected as the operating system.

5.2 Selection of the Medium and Tools

Linux distribution of SUSE has a configuration utility called Yast. It is easy to manage, setup and install applications with a graphical interface for the operating system. Users are added and passwords are assigned for these users via Yast and these accounts became the target of the cracker during the testing phase.

Process Monitoring and Management window is also a very useful application for the Linux users. Such programs let the user to monitor, manage, hang up, set the level of nicing, wait, and kill the processes running on the system. Nicing option for a process can be used if a process takes too much CPU time and forces the other processes to fall into dead lock and/or live lock phases. Meanwhile, the cracker can run for a long time without affecting the common jobs of the server with a low priority level. An administrator can let the cracker work for weeks if the etc/passwd file has to many user accounts or the wordlist is too long.

Table 5.1 and 5.2 summaries the hardware and software profiles selected, respectively.

Table 5.1 Hardware Profile.
Intel Pentium Celeron II
233 MHz CPU
32 MB SDRAM
6.2 GB Hard Disk

Table 5.2 Software Profile.
SUSE 6.2
Kernel 2.2.10
KDE 1.1.1
GCC, GNU project compiler (egcs-1.1.2)
Library for C compiler (v.2.1.1-9)

5.3 Implementation of the Program

Sub-programs Coded:

Several sub-programs with read and write operations are coded and unit tests are prepared. String commands are used to compare words, to eliminate carriage-returns from strings, and append two words.

In order to keep the privacy of the users, passwords can be unstringed from the user line of the password file. Only the password hashes containing 13 characters can be taken out of the file and the cracker work on them. In such a case, user name as a word will not be utilized to crack the password.

A sub-program is coded to generate all possible salting values. This can be used either when the salt value of the target is not known or if a precomputed dictionary attack is being used. 4096 possible values are thus generated and used as input for a call to crypt() as shown in Figure 5.1 below.

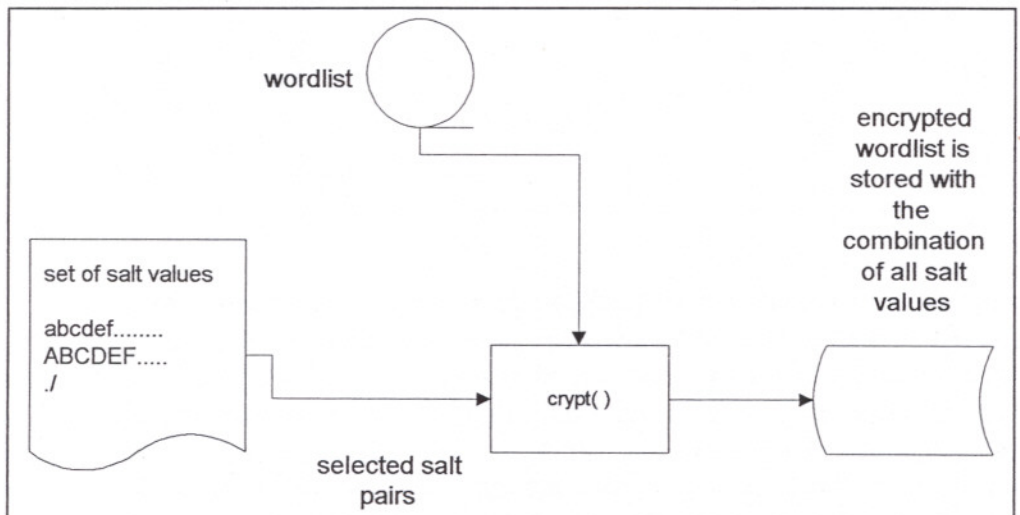

Figure 5.1 Salt Generation.

As shown in Figure 5.2 below, the main algorithm is being used to read a hash from the etc/passwd file, and compare it with the hash value of the encrypted object. Here, the hash value of object is the output of the crypt() with the inputs; sequentially read wordlist and the salt taken from the 2 bytes of the password in the etc/passwd file. These approaches have been taken under the circumstances that already defined in section 4.5. Therefore the salt is known if the password file is available.

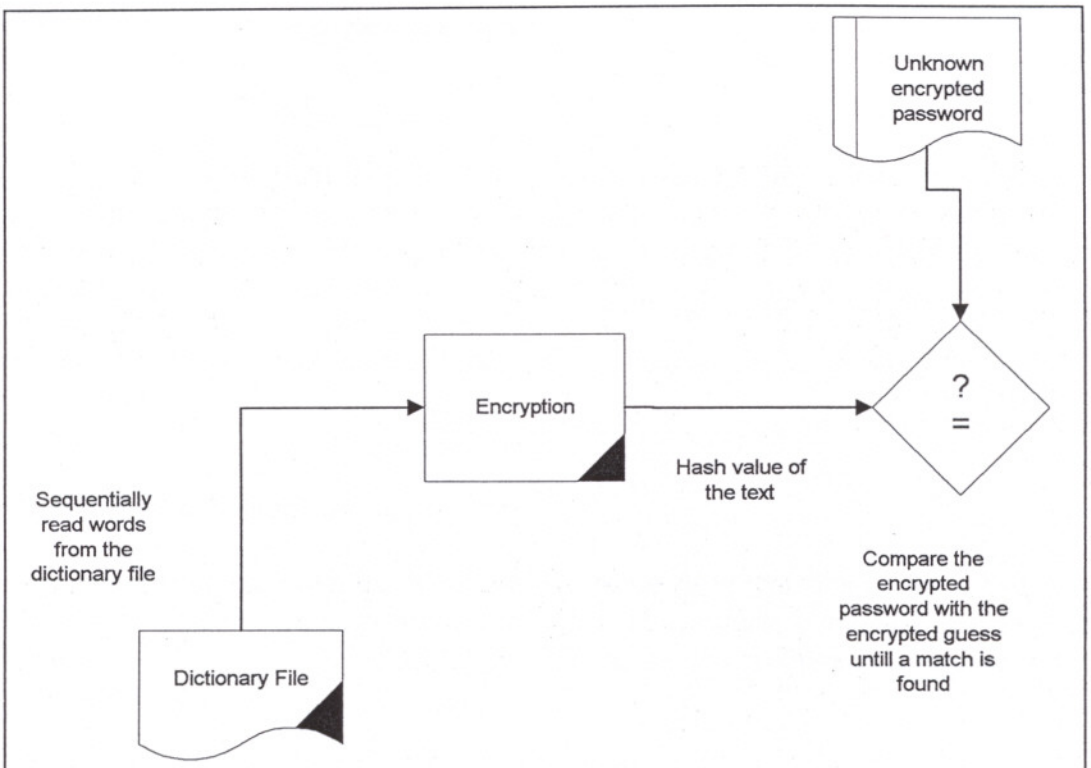

Figure 5.2 Password Cracking Process.

In order to keep the run-time of the program shorter, some of the data are kept in memory for less I/O operations. Both the password file and the dictionary are input files and more over the dictionary file is to be rewinded for each hash in the password file. Now, this causes more I/O operation, therefore a copy of the wordlist is loaded into the memory as the initial section of the program. Such manipulations and command elimination are used during the coding phase. There will always be alternatives for runtime optimization; however, the program can be regarded as acceptable as it is.

Dynamic memory allocation is used for tables to obtain more memory space at execution time to hold new nodes, and to release space no longer needed. Functions *malloc* and *free*, and the operator *size of*, are essential to dynamic memory allocation. Function malloc takes as an argument the number of bytes to be allocated, and returns a pointer of type void* to the allocated memory. A void* pointer may be assigned to a variable of any pointer type.[1] With the statement

newPtr = malloc(size of (typeName));

the memory is allocated on the heap (the extra memory available to the program at execution time).[2]

---

[1] H. M. Deitel, P. J. Deitel, *C How to Program,* p. 470.

[2] H. M. Deitel, P. J. Deitel, *C How to Program,* p. 576.

The free function de-allocates memory,

Free(newPtr);

Once a program is coded it is compiled and an output file is created by the command below. The –o option applies regardless to whatever sort of output GCC is producing, whether it be an executable file, an object file, an assembly file, or a preprocessed C code.

$ gcc <source.c> < link edited output> –o

and the

$ ./<link edited output >

command runs the program. Since we are using the crypt library for a call to the function crypt() which is shown in Figure 5.3. The compile command above will not be enough to link the glibc-crypt library. Therefore, the program is compiled with the command;

$ gcc <source.c> <object file name> –o –lcrypt

```
# define _XOPEN_SOURCE
# include <unistd.h>
char *crypt(const char *key, constant char *salt);


key: user password
salt: [ a...zA...A0...9./ ]
```

Figure 5.3 Call to crypt().

If the password hashes can not be cracked by the first (simple mode) method then the numbering mode (appendAge mode) is activated. In this mode, numbers form 40 to 99 are appended to the end of each word in the dictionary in an incremented way. Some passwords are expected to have two digits stringed at the end of a word or a name. Some people use their passwords just the same of their user names with the year of birth at the end. This is one of the common mistakes done, although the user mistakenly thinks no one can either guess or know the birth year of him but that it is not much hard for the attacker to work on a set of possible digits.

5.4 Results

The cracker -John the Ripper- was ran on an etc/passwd file and the successful matches obtained in the first 3 hours which are:

Men/women names: 2.6 %
Several patterns: 0.3 %

The coded application is also executed for the same etc/passwd file. Two experiments are done. In the first one a dictionary containing 3723 men and women names is used. In the second experiment a longer wordlist that contains 38595 words is used. This long wordlist is obtained by a combination of several online wordlists by eliminating the doubles. One thing that must be mentioned here is that there is a small difference between a wordlist and a dictionary and, thus crypt() uses the left most eight characters of an input word. Therefore using two different words whose first eight characters are the same as an input for the crypt() respectively, gives the same output hash value. Before using a dictionary for the cracker, the dictionaries are combined and truncated starting from the 9^{th} byte. Then the doubles are eliminated. This procedure supplies an effective wordlist that can be used during an optimum execution time.

The results of the two experiments are given in the Tables 5.3 and 5.4 respectively. The union of the results of these two experiments gives us a successful match of 6.31 %.

Table 5.3 Results of Experiment #1.

Real execution time	**: 01:29:30 (hh:mm:ss)**
Total number of passwords	**: 301**
Total number of cracked passwords	**: 12**
Percentage of cracked passwords	**: 3.99 %**
by simple mode	**: 0.03 %**
by AppendAge mode	**: 3.96 %**
Password Content	
Women's names	**: 1.66 %**
Men's names	**: 2.33 %**

Table 5.4 Results of Experiment #2.

Real execution time	**: 17:56:15 (hh:mm:ss)**
Total number of passwords	**: 301**
Total number of cracked passwords	**: 10**
Percentage of cracked passwords	**: 3.32 %**
by simple mode	**: 0.66 %**
by AppendAge mode	**: 2.65 %**
Password Content	
Places	**: 0.33 %**
Job names	**: 0.33 %**
Favorite meal names	**: 0.33 %**
Phrases and patterns	**: 2.33 %**

The application was designed and coded to work as batch program. It uses an output file to report and log the actions. It reports the real time and the CPU clock as soon as it starts and opens the files, writes the cracked passwords, number of words in the dictionary, number of cracked passwords, and any other predefines cases into this file. At the end of the execution just before the internal controls like closing the files, it again reports the time. Therefore, the user can run and leave the job, he then may claim the results by browsing the *out.dat* file later.

The security of each individual user whose passwords have been cracked is closely related to the security of the whole system. The owner of the classified passwords above often have no idea how a multi-user system works. Their poor choices opens a door for an outsider to the entire system. The cracked passwords are all in the set of bad passwords that are described in section 4.6.1. They consist of strings that are easy to remember for their owners and if there is a set of numbers appended to the end of this word it is usually the year of birth or the year that the account is activated.

CHAPTER 6

CONCLUSIONS

6.1 Concluding Remarks

In this study, it is described that some of the passwords in an UNIX based operating system can be cracked by an automated dictionary attack. As described earlier, the security of the entire system can be at risk if the user password security is in risk. The results of the experiments given in the Chapter 5 show that the users of a UNIX based operating system must be educated about choosing and using their passwords due to fact that the cracked passwords were all in the set of weak passwords. Meanwhile, it is obvious that the number of cracked passwords can be increased by using a longer wordlist and moreover the amount of execution time can be decreased by running the program on a computer that has a faster CPU.

System administrators must ensure that all the passwords in the system are strong which means that; they are all well chosen. This can be supplied by checking the passwords when they are first created or by regular checking the strength of the passwords in the operating system.

6.2 Suggestions for Future Work

The applied cracker program can be developed by applying new sections. An attack with a better wordlist will always give better results hence a higher percentage of cracked passwords.

The dictionaries can be classified by their categories before they are used in cracking. Therefore the result of the attack will give better ideas about the passwords in the etc/passwd file.

The letters of words in the dictionaries can be capitalized; this can be applied for all of the letters or only for the first and the last letters of a word. A digit, a punctuation mark, an up arrow, or a space can be appended to the end of each word in the wordlist.

SUMMARY

In this study, it is aimed to crack the password hashes which are encrypted by DES, in an UNIX-based operating system using the method of dictionary attack. The developed application is introduced.

Remarks and recommendations are given for the users by defining the good and bad password choosing methods. Secure methods for user authentication, user management and for password storage are introduced for the system administrators to let them check the security of user accounts in the system dynamically and frequently.

The results of the developed program shows us the threat of weak password choices over the entire operating system. For future studies; the utility can be developed to give a higher percentage of cracked passwords in a shorter running time. Recommendations are given about software and hardware requirements that must be taken into account for the future study.

ÖZET

Bu çalışmada, DES algoritması ile şifrelenmiş UNIX tabanlı işletim sistemi kullanıcı şifrelerinin sözlükten tarama ve saldırı yöntemiyle kırılması amaçlanmış ve geliştirilen uygulama tanıtılmıştır.

İyi ve kötü seçilmiş şifre tanımlamaları yapılarak, kullanıcılara şifrelerini özenle seçmeleri için bazı uyarı ve tavsiyelerde bulunulmuştur. UNIX tabanlı işletim sistemlerinin kullanıcı parolalarını şifreleme ve saklama teknikleri tanıtılmış böylece sistem yöneticilerinin kullanıcı hesaplarını ve sistem güvenliğini aktif olarak izlemeleri sağlanmıştır.

Geliştirilen uygulamanın sonuçlarından yola çıkarak kullanıcıların zayıf şifre seçimlerinin tüm sistemin güvenliğini olumsuz yönde etkiledikleri gözlemlenmiştir. İlerideki çalışmalar için yazılım ve donanım gereksinimleri göze alınarak, uygulamanın geliştirilmesi ve daha kısa çalışma süresi ile daha yüksek bir şifre kırma yüzdesine ulaşmak mümkündür. Bunun için tavsiyelerde bulunulmuştur.

BIBLIOGRAPHY

ANONYMOUS, *Maximum Linux Security,* Sams Publishing, 2000.

BELGERS, Walter, "UNIX Password Security", December 6, 1993.

CONS, Lionel, *CERN Security Handbook Practical Computer Security for CERN Users Version 1.2,* 12 December 1996.

CURRY, David A., *Improving The Security of Your UNIX System,* Information and Telecommunications Sciences and Technology Division, ITSTD-721-FR-90-21.

DEITEL, H. M., DEITEL, P. J., *C How to Program,* 2nd Edition, Prentice Hall, 1994.

FELDMEIER, D.C., KARN P.R., "UNIX Password Security- Ten Years Later".

FIPS, Federal Information Processing Standards Publication, "Data Encryption Standard, Fips Pub 46-3", Reaffirmed 1999 October 25 U.S. Department of Commerce/National Institute of Standards and Technology.

FIPS, Federal Information Processing Standards Publication 81, "Announcing the Standard for Des Modes of Operation", 1980 December 2.

GARFINKEL, S., SPAFFORD, G., *Practical UNIX Security,* O'Reilly and Associates, Inc., 1991.

GONG, Li, "Protecting Poorly Chosen Secrets from Guessing Attacks".

JOHN THE RIPPER, Password Cracker Manual, Internet Document, http://www.openwall.com/john, 2001.

L0PHTCRACK, Password Cracker v.2.5 Manual, Internet Document, Internet Document, http://www.l0pht.com, 2000.

MORRIS, Robert, THOMPSON, K., "Password Security: A Case History".

MUFFETT, Alec E. "Almost Everything You Wanted To Know About Security (but were afraid to ask!)", Internet Document, USENET newsgroup.alt.security.

MUFFETT, Alec, "Crack Version v5.0 User Manual", Internet Document, http://www.users.dircon.co.uk/~crypto/, 2001.

PARKER, Tim, "Linux System Administrator's Survival Guide", Internet Document, http://10.10.0.5/htmlbooks/, May 2001.

PFLEEGER, Charles P., *Security in Computing,* Prentice Hall, 1989.

SAMS DEVELOPMENT TEAM (PARKER, Scott, SMITH, Jeff, et.al., *UNIX Unleased*, Sams Publishing, 1994.

SHANNON, Claude, "Prediction and Entropy of Printed English", Bell System Technical Journal, 30(1):50-64, January 1951.

STEINER, J.G., NEUMAN, C., SCHILLER, J.I., "Kerbelos, An Authentication Service for Open Network Systems", USENIX Conference Proceedings, Dallas, Texas, February 1988.

VENEMA, Wietse, "Using SecurID Tokens in an Open Multi-host UNIX Environment", 1993.

WOOD, P.H., KOCHAN, S.G., *UNIX System Security,* Hayden Books. ISBN 0-8104-6267-2. 1985.