8.6 The UNIX Encrypted Password System

When UNIX requests your password, it needs some way of determining that the password you type is the correct one. Many early computer systems (and quite a few still around today!) kept the passwords for all of their accounts plainly visible in a so-called "password file" that really contained the passwords. Under normal circumstances, the system protected the passwords so that they could be accessed only by privileged users and operating system utilities. But through accident, programming error, or deliberate act, the contents of the password file almost invariably become available to unprivileged users. This scenario is illustrated in the following remembrance:

Perhaps the most memorable such occasion occurred in the early 1960s when a system administrator on the CTSSsystem at MIT was editing the password file and another system administrator was editing the daily message that is printed on everyone's terminal on login. Due to a software design error, the temporary editor files of the two users were interchanged and thus, for a time, the password file was printed on every terminal when it was logged in. - Robert Morris and Ken Thompson, Password Security: A Case History

The real danger posed by such systems, wrote Morris and Thompson, is not that software problems will cause a recurrence of this event, but that people can make copies of the password file and purloin them without the knowledge of the system administrator. For example, if the password file is saved on backup tapes, then those backups must be kept in a physically secure place. If a backup tape is stolen, then *everybody's* password must be changed.

UNIX avoids this problem by not keeping actual passwords anywhere on the system. Instead, UNIX stores a value that is generated by using the password to encrypt a block of zero bits with a one-way function called *crypt( )* ; the result of the calculation is (usually) stored in the file */etc/passwd* . When you try to log in, the program */ bin/login* does not actually decrypt your password. Instead,*/bin/login* takes the password that you typed, uses it to transform another block of zeros, and compares the newly transformed block with the block stored in the */etc/passwd* file. If the two encrypted results match, the system lets you in.

The security of this approach rests upon the strength of the encryption algorithm and the difficulty of guessing the user's password. To date, the *crypt* () algorithm has proven highly resistant to attacks. Unfortunately, users have a habit of picking easy-to-guess passwords (see Section 3.6.1, "Bad Passwords: Open Doors"), which creates the need for shadow password files.

NOTE: Don't confuse the *crypt ( )* algorithm with the *crypt* encryption program. The *crypt* program uses a different encryption system from *crypt ( )* and is very easy to break. See Chapter 6, *Cryptography* , for more details.

8.6.1 The crypt() Algorithm

The algorithm that *crypt ( )* uses is based on the Data Encryption Standard (DES) of the National Institute of Standards and Technology (NIST). In normal operation, DES uses a 56-bit key (eight 7-bit ASCII characters, for instance) to encrypt blocks of original text, or *clear text* , that are 64 bits in length. The resulting 64-bit blocks of encrypted text, or *ciphertext* , cannot easily be decrypted to the original clear text without knowing the original 56-bit key.

The UNIX *crypt ( )* function takes the user's password as the encryption key and uses it to encrypt a 64-bit block of zeros. The resulting 64-bit block of cipher text is then encrypted again with the user's password; the process is repeated a total of 25 times. The final 64 bits are unpacked into a string of 11 printable characters that are stored in the */etc/passwd* file.[8]

[8] Each of the 11 characters holds six bits of the result, represented as one of 64 characters in the set ".", "/", 0-9, A-Z, a-z, in that order. Thus, the value 0 is represented as ".", and 32 is the letter "U".

Although the source code to *crypt ( )* is readily available, no technique has been discovered (and publicized) to translate the encrypted password back into the original password. Such reverse translation may not even be possible. As a result, the only known way to defeat UNIX password security is via a brute-force attack (see the note below), or by a *dictionary attack* . A dictionary attack is conducted by choosing likely passwords, as from a dictionary, encrypting them, and comparing the results with the value stored in*/etc/passwd* . This approach to breaking a cryptographic cipher is also called a *key search* or *password cracking.*

Robert Morris and Ken Thompson designed *crypt ( )* to make a key search computationally expensive, and therefore too difficult to be successful. At the time, software implementations of DES were usually slow; iterating the encryption process 25 times made the process of encrypting a single password 25 times slower still. On the original PDP -11 processors, upon which UNIX was designed, nearly a full second of computer time was required to encrypt a single password. To eliminate the possibility of using DES hardware encryption chips, which were a thousand times faster than software running on a PDP -11, Morris and Thompson modified the DEStables used by their software implementation, rendering the two incompatible. The same modification also served to prevent a bad guy from simply pre-encrypting an entire dictionary and storing it.

What was the modification? Morris and Thompson added a bit of *salt* , as we'll describe below.

NOTE: There is no published or known method to easily decrypt DES -encrypted text without knowing the key.[9] However, there have been many advances in

hardware design since the DES was developed. Although there is no known software algorithm to "break" the encryption, you can build a highly parallel, special-purpose DES decryption engine that can try all possible keys in a matter of hours.

The cost of such a machine is estimated at several millions of dollars. It would work by using a brute-force attack of trying all possible keys until intelligible text is produced. Several million dollars is well within the budget of most governments, and a significant number of large corporations. A similar machine for finding UNIX passwords is feasible. Thus, passwords should not be considered as completely "unbreakable."

8.6.2 What Is Salt?

As table salt adds zest to popcorn, the salt that Morris and Thompson sprinkled into the DES algorithm added a little more spice and variety. The DES salt is a 12-bit number, between 0 and 4095, which slightly changes the result of the DES function. Each of the 4096 different salts makes a password encrypt a different way.

When you change your password, the */bin/passwd* program selects a salt based on the time of day. The salt is converted into a two-character string and is stored in the */etc/passwd* file along with the encrypted "password."[10] In this manner, when you type your password at login time, the same salt is used again. UNIX stores the salt as the first two characters of the encrypted password.

[10] By now, you know that what is stored in the */etc/passwd* file is not really the encrypted password. However, everyone calls it that, and we will do the same from here on. Otherwise, we'll need to keep typing "the superencrypted block of zeros that is used to verify the user's password" everywhere in the book, filling many extra pages and contributing to the premature demise of yet more trees.

Table 8.2 shows how a few different words encrypt with different salts.

Table 8.2: Passwords and Salts

Password	Salt	Encrypted Password
nutmeg	Mi	MiqkFWCm1fNJI
ellen1	ri	ri79KNd7V6.Sk
Sharon	./	./2aN7ysff3qM
norahs	am	amfIADT2iqjAf
norahs	7a	7azfT5tIdyh0I

Notice that the last password, *norahs* , was encrypted two different ways with two different salts.

Having a salt means that the same password can encrypt in 4096 different ways. This makes it much harder for an attacker to build a reverse dictionary for translated encrypted passwords back into their unencrypted form: to build a reverse dictionary of 100,000 words, an attacker would need to have 409,600,000 entries. As a side effect, the salt makes it possible for a user to have the same password on a number of different computers and to keep this fact a secret (usually), even from somebody who has access to the*/etc/passwd* files on all of those computers; two systems would not likely assign the same salt to the user, thus ensuring that the encrypted password field is different.[11]

[11] This case occurs only when the user actually types in his or her password on the second computer. Unfortunately, in practice system administrators commonly cut and paste */etc/passwd* entries from one computer to another when they build accounts for users on new computers. As a result, others can easily tell when a user has the same password on more than one system.

8.6.3 What the Salt Doesn't Do

Unfortunately, salt is not a cure-all. Although it makes the attacker's job of building a database of all encrypted passwords more difficult, it doesn't increase the amount of time required to search for a single user's password.

Another problem with the salt is that it is limited, by design, to one of 4096 different possibilities. In the 20 years since passwords have been salted, computers have become faster, hard disks have become bigger, and you can now put 4, 10, or even 20 gigabytes of information onto a single tape drive. As a result, password files have become once again a point of vulnerability, and UNIX vendors are increasingly turning to shadow password files and other techniques to fight password-guessing attacks. Yet another problem is that the salt is selected based on the time of day, which makes some salts more likely than others.

8.6.4 Crypt16() and Other Algorithms

Some UNIX operating systems, such as HP-UX, Ultrix, and BSD 4.4 can be configured to use a different *crypt ( )* system library that uses 16 or more significant characters in each password. The algorithm may also use a significantly larger salt. This algorithm is often referred to as *bigcrypt* () or *crypt16* (). You should check your user documentation to see if this algorithm is an option available on your system. If so, you should consider using it. The advantage is that these systems will have more secure passwords. The disadvantage is that the encrypted passwords on these systems will not be compatible with the encrypted passwords on other systems.