CS415 Module 10 Part B - Users and Security

Athens State University

Outline

User Management in Classic UNIX

- In Classic UNIX, there are two types of users: a single omnipotent administrator user and all other users
- Each user is assigned a unique login name and associated numeric user identifier (UID)
- Users are assigned to one or more groups with each group having a unique name and numeric group identifier (GID)

The Password File: /etc/passwd

```
1 alewis:x:1000:100:Adam Lewis:/home/alewis:/bin/bash
```

- A system file that containing one line per user on the system
- Text file, with each record separated by colons
- Data is used by multiple subsystems, and must be user readable

The Shadow Password File: `/etc/shadow`

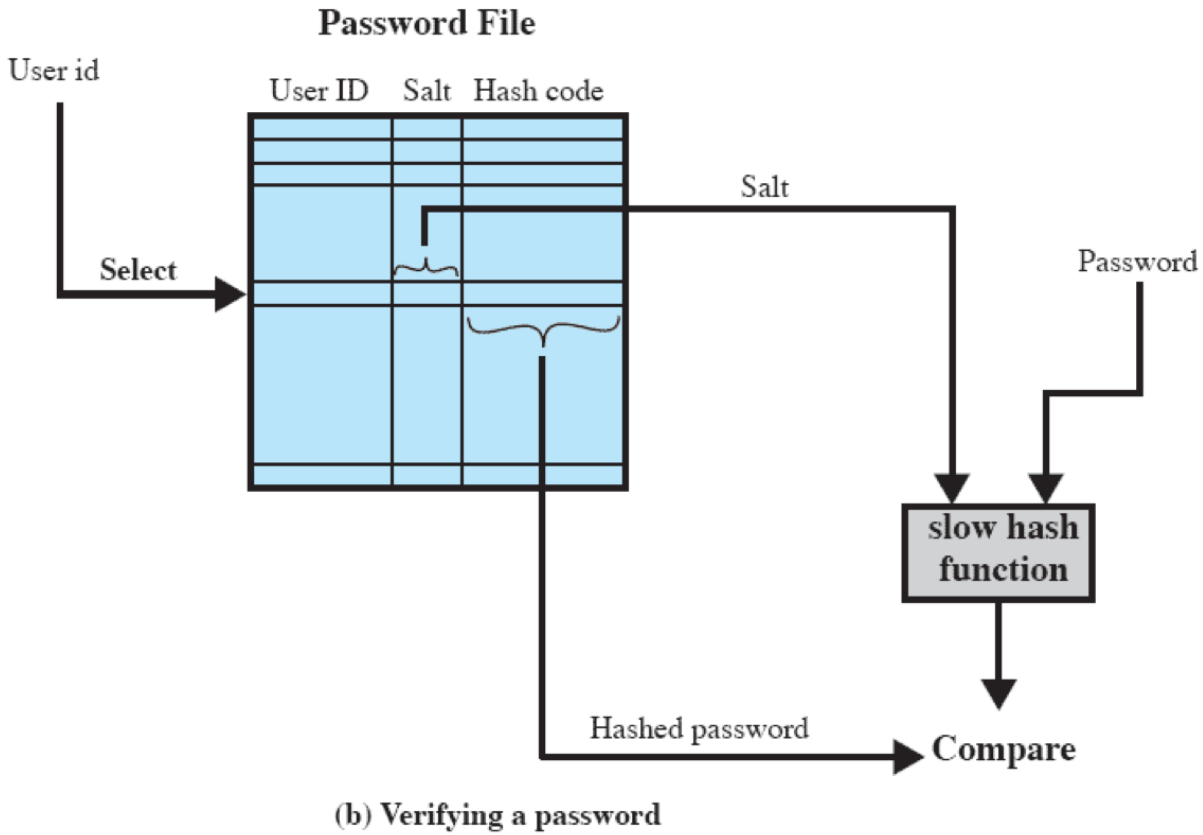
- Early modern versions of UNIX (SysV, BSD4.4, and others) introduced the concept of password shadowing
- The password file exposes too much information to unprivileged system utilities
- So, all of sensitive information about users was moved into the shadow file
  - Access to said file was limited to administrative users

The Groups File: /etc/groups

```
1 users:x:100:
  faculty:x:106:alewis, kmayfield, bradyrimes, dave,
      dhill
3 students:x:107:
  adm:x:108:alewis,dave
5 print:x:109:alewis,kmayfield,bradyrimes,dave,dhill
```

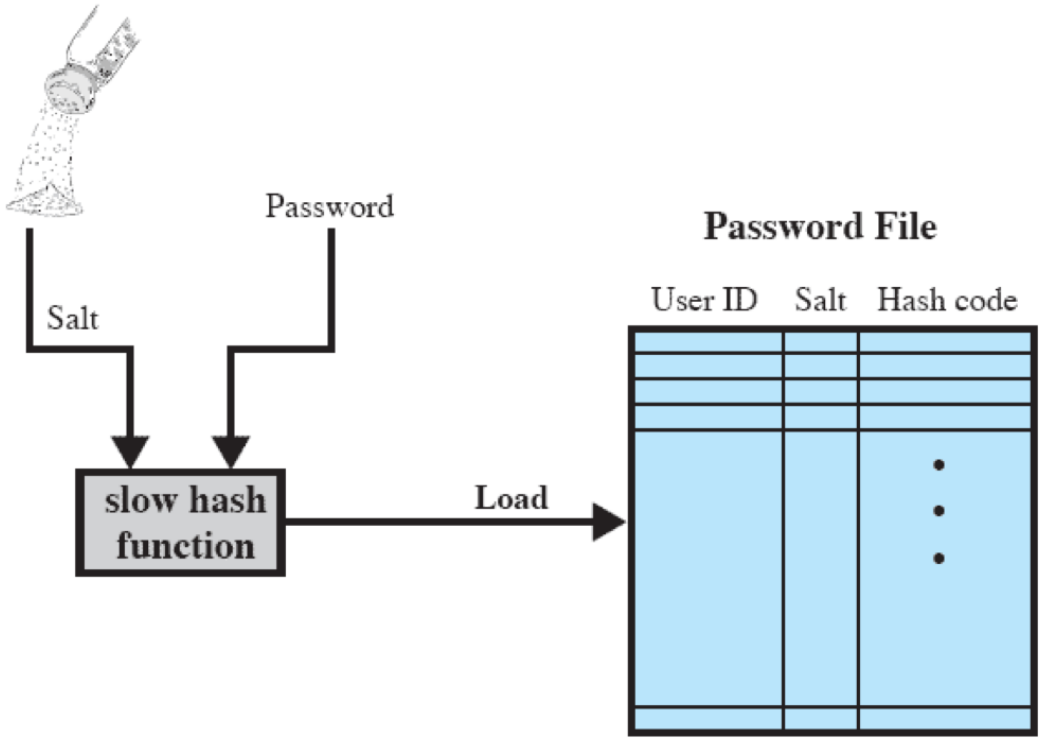
- File contains one line for each group in the system
- Text file, with fields separate by colon

UNIX Password Verification



(b) Verifying a password

Hashing and Salting of Passwords



(a) Loading a new password

The Purposes of Salt

1. Prevents duplication passwords from being visible in the shadow file
   - Even if two users choose the same password, the passwords will be assigned different salt values
2. Greatly increases the difficulty of offline dictionary attacks
3. It becomes nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them

Password Encryption and Authentication

- Many applications have to provide some form of login to a remote system (`ssh` or `ftp`, for example)
- UNIX systems encrypt passwords using *one-way encryption*
  - Means that we re-create a password given it's encrypted form
  - Result is that we have to validate a password by encrypting it and see if the encrypted value matches the encrypted value in the shadow file

The crypt() System Call

```
1 #include <unistd.h>
  char *crypt(const char *key, const char *salt)
```

- This system call supports MD5 (preferred) or DES hashing
- The value of `salt` selects the algorithm
- For a new password, salt should be set to a reasonably random value

Example: Hashing a password using `crypt()`

```c
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <crypt.h>
int main() {
  unsigned long seed[2];
  char salt[] = "$1$........";
  const char *seedchars = "./0123456789ABCDEFGHIJKLMOPQRSTUVWXYZ";
  char *password = NULL;
  /* This is not a very good seed */
  seed[0] = time(NULL);
  seed[1] = getpid() ^ (seed[0] >> 14 & 0x30000);
  /* Make it printable using chars from seedchars */
  for (int i = 0; i < 8; i++) {
    salt[3+i] = seedchars[(seed[i/5] >> (i%5)*6) & 0x3f];
  }
  /* Ask user to enter a password and encrypt it */
  password = crypt(getpass("Password:"), salt);
  /* Print the result */
  puts(password);
  return 0;
}
```

Breaking `crypt()`

- There are two threats to this scheme:
  - A user can gain access to a machine using a guest account
  - Or they can use a *password cracker* such as John the Ripper to guess a password based on a hash
- If an attacker can get a copy of the password file, then a password cracker can run on a different machine to break these passwords

Process Credentials

- Every process is associated with a set of UIDs and GIDs known as **process credentials**
  - Real UID and GID
  - Effective UID and GID
  - Saved set-user-ID and saved set-group-ID
  - File system UID and GID
  - Supplementary GIDs

Effective UID and GID

- In Classical UNIX, the effective UID and GID determine the permissions granted to a process when it tries to do things
- A *privileged process* is a process whose effective UID has been set to 0
- A process can adjust its eUID and eGID either by command (i.e. `sudo` or `su`) or via system call

Saved Set-User-ID and Saved Set-Group-ID

- When a program executes, two things happen (among many things):
  1. If the program executable file has the set-user-ID (set-group-ID) bit set, then the effective user of the process is set to the owner of the executable file
  2. The values for the saved set-user-ID and saved set-group-ID are copied from the corresponding effective IDs
- System calls exist that allow a process running a set-user-ID program to switch its effective user ID from it's real ID to saved-set-UID and vice-versa ID to its

Supplementary Group IDs

- A process can belong to more than one group, as defined in the `/etc/group` file
- These GIDs are used with the effective GID to determine what permissions have been granted to a file