Authentication

(1) Authentication Basics

- Authentication is the binding of an identity to a subject
- Authentication Methods:
  - What you know (such as passwords)
  - What you have (such as a badge)
  - Who you are (fingerprints)
  - Where you are (such as in front of a particular terminal)

---

(2) Password Authentication in Early UNIX

- Password
  - Password is at most 8 bytes long
  - 128 (= 2^7) possible characters
  - 95 different printable characters
  - Search space 2^{53}.

- How to store passwords
  - Should we store passwords in plaintext?
    - Definitely not!
    - Then how do we verify whether a user has typed a correct password?
  - Early Unix
    - Takes password as the encryption key
    - Use it to encrypt a 64-bit block of zeros
    - The resulting 64-bit block of cipher text is then encrypted again with the password
    - The whole 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. (each of the 11 characters holds 6 bits, represented as one of 64 characters in the set ".", "/", 0-9, A-Z, a-z, in that order)
  - More general solution
    - The one-way hash of the password is stored (such as MD5 or SHA1).

- Salt
  - From the password file, I might be able to know whether somebody has the same password as mine.
  - Dictionary attack on the entire password file:
    - Generate the one-way hash for all the words in a dictionary
    - Check whether anybody's password hash matches with any string of the hashed dictionary.
  - How to solve the above two problems?
    - Making the one-way hashes of passwords different even if they are the same
  - Using "Salt":
    - Attach a random string before each password;
    - The random string should be public and stored in the password file.

- The `/etc/passwd` file
  - Format: username:password:uid:gid:full_name:home_dir:shell
  - Example: wedu:-----:31121:31000:Wenliang Du:/home/cisfac/wedu:/bin/tcsh
  - Sun's NIS: to get the real passwd, using `ypcat passwd`

- The shadow Password.
  - `/etc/passwd` is world-readable. Any user can get the encrypted passwords, and conduct password cracking attacks.
  - Why not `chmod /etc/passwd` to 600 and make it world unreadable?
    - Many programs need to consult it for UID to account name mappings, home directories, and username information.
  - The encrypted password is moved to `/etc/shadow`.
    - The password entry in `/etc/passwd` is replaced with an "`x`"
    - Example: wedu:x:31121:31000:Wenliang Du:/home/cisfac/wedu:/bin/tcsh
  - Format: `username:passwd:last:may:must:warn:expire:disable:reserved`
    - last: Days since Jan 1, 1970 that password was last changed
    - may: Days before password may be changed
    - must: Days after which password must be changed
    - warn: Days before password is to expire that user is warned
    - expire: Days after password expires that account is disabled
    - disable: Days since Jan 1, 1970 that account is disabled
    - reserved: A reserved field

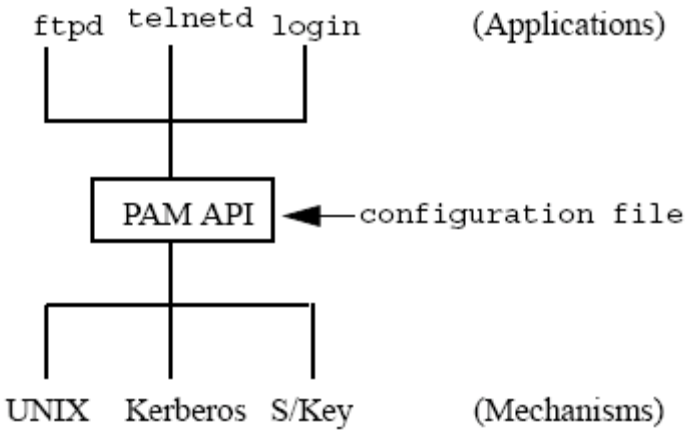
- How to Protect Passwords?
  - Dictionary attack
    - Klein's study: 8% dictionary words, 4% common names, 3% account names, etc.
  - Limit login attempts
    - Blocking login attempts after 3 failed trials?
    - A potential Denial of Service attack.
  - Proactive password checker
  - Password aging

- Login Spoofing Attack
  - What is login spoofing?
  - The ATM machine spoofing story
  - How to prevent the spoofing attack?
    - Displaying the number of failed logins
    - Trusted path: CTRL + ALT + DEL
    - Mutual authentication: user also authenticates the machine.
  - Another example: Risk of running "`su`" or "`ssh`" from other people's account.
    - use `/usr/bin/su` instead of "`su`"

- Other potential risks
  - Passwords are stored in memory, cache for long time.
  - Zero-out the memory

(3) PAM: Pluggable Authentication Module

Principle of Flexibility: There is no free lunch in security. Better security usually comes with a price, and how much price users want to pay is up to users, not up to the developers. Therefore, if possible, we should give users a choice of security mechanisms, rather than choosing it for them and hardwiring a security mechanism in the implementation. In other words, we should make our system flexible to accommodate multiple security mechanisms. PAM is an excellent application of this principle. In this lecture, it is not an intention to teach how PAM works, because they change from versions to versions; instead, we would like to teach students the principle of flexibility, and we use PAM as an example to illustrate this principle.



- Motivation:
  - Separating applications from authentication, and thus making authentication independent.
  - Authentication mechanisms are not hard-coded in applications.
  - We can configure what authentication mechanisms to use for an application.
  - PAM were first developed by Sun Microsystems.
  - PAM is currently supported by AIX, HP-UX, Solaris, Linux, FreeBSD, Mac OS X and NetBSD.

- Linux-PAM
  - The directory `/etc/pam.d/` contains the PAM configuration files for each PAM-aware applications (in early versions of PAM, the file `/etc/pam.conf` was used for all applications).
  - PAM-aware applications use the standard PAM APIs to invoke authentication modules. PAM depends on the configuration file to decide which module to invoke.
  - Stacking modules: we can stack several authentication modules together, so that multiple modules are used together for authentication. The order in which the modules are listed is very important.

```
auth       required     /lib/security/pam_nologin.so
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_env.so
auth       sufficient   /lib/security/pam_rhosts_auth.so
auth       required     /lib/security/pam_stack.so service=system-auth
```

- In addition to authentication management, PAM also provides password management, account management, and session management.

❖ Example:
- Configuration file: `/etc/pam.d/check_user`

```
#%pam-1.0
auth       required   pam_unix_auth.so
account    required   pam_unix_acc.so
```

- Example: Invoking PAM modules

```
retval = pam_start("check_user", user, &conv, &pamh);

if (retval == PAM_SUCCESS) {
    retval = pam_authenticate(pamh, 0);    /* is user really user? */
}
if (retval == PAM_SUCCESS){
    retval = pam_acct_mgmt(pamh, 0);       /* permitted access? */
}

/* This is where we have been authorized or not. */
if (retval == PAM_SUCCESS) { fprintf(stdout, "Authenticated\n"); }
else { fprintf(stdout, "Not Authenticated\n"); }

if (pam_end(pamh,retval) != PAM_SUCCESS) {     /* close Linux-PAM */
      pamh = NULL;
      fprintf(stderr, "check_user: failed to release authenticator\n");
      exit(1);
}
```

(4) Account Management

- Accounts that run a single command
  `date::60000:100:Run the date program:/tmp:/usr/bin/date`

- `chsh:` changing the account's login shell.
- `chfn:` change the finger information
  - An existing vulnerability: allow any user to become root.
  - It allows user to add the following line to `/etc/passwd`
    - `xyz::0:0:::`
    - You have created a root account (uid = 0).

- Restricting Logins:
  - How can we restrict that any login to an account can only occur from 8am to 5pm?
  - Write a script, and use "chsh" to put it in `/etc/passwd`
  - PAM provides a more generic solution
    - `pam_nologin.so:` does not allow login
    - `pam_securetty.so:` does not allow anyone to log in remotely as a root user over an uncrypted network connection

(5) Authentication Over Network

- **Network Information Service (NIS)**
  - When you're running a local area network, your overall goal is usually to provide an environment for your users that makes the network transparent. An important stepping stone is keeping vital data such as user account information synchronized among all hosts. This provides users with the freedom to move from machine to machine without the inconvenience of having to remember different passwords and copy data from one machine to another. Data that is centrally stored doesn't need to be replicated, so long as there is some convenient means of accessing it from a network-connected host. By storing important administrative information centrally, you can make ensure consistency of that data, increase flexibility for the users by allowing them to move from host to host in a transparent way, and make the system administrator's life much easier by maintaining a single copy of information to maintain when required.
  - The NIS server program is traditionally called **`ypserv.`**
  - How does a client find out which NIS server to connect to
    - Static configuration
    - Dynamic discovery: **`ypbind.`**
  - NIS commands
    - `%ypcat passwd` (Maps encrypted passwords to user login names)
    - `%ypcat group` (Maps Group IDs to group names)

- **Kerberos**
  - A network authentication protocol developed by MIT.
  - Provide strong authentication for client/server applications
  - A client can prove its identity to a server (and vice versa) across an *insecure* network connection.

- Kerberos builds on symmetric key cryptography and requires a trusted third party.
- Kerberos are used by
  - Windows 2000, Windows XP and Windows Server 2003
  - Apple's Mac OS X
  - OpenSSH
  - Network File System (NFS) from Sun Microsystems

- Many other authentication protocols.
  - Crypto protocols
  - One-time password, SKEY.
  - Authentication using public key cryptography.