User Authentication

File /etc/passwd

- Maps user names to user ids (many applications legitimately need this)
- No legitimate need for encrypted passwords

File /etc/shadow

- Contains salted & hashed passwords
- Account information (last change, expiration)
- Readable only by superuser and privileged programs
- Different hash algorithms supported
  - DES
  - MD5
  - SHA-{256,512}

DEMO
passwd vs. shadow

Unix Groups

Users belong to one or more groups

- Primary group (stored in `/etc/passwd`)
- Additional groups (stored in `/etc/group`)
- Possibility to set group password
- Become group member with `newgrp`

File /etc/group

```
groupname : password : group id : additional users
root:x:0:root
bin:x:1:root,bin,daemon
users:x:1000:pizzaman
```

Special group wheel

- Group for users that can call `su`

DEMO

id

File System

Directory tree

- Primary repository of information
- Hierarchical set of directories
- Directories contain file system objects (FSO)
- Root is denoted as "/"

File system objects (FSO)

- Files, directories, symlinks, sockets, device files
- FSOs Have names but are really referenced by inode (index node)

File System

- Access Control
  - Permission bits
  - chmod, chown, chgrp, umask
  - File listing

```
    -         rwx    rwx        rwx
(file type) (user) (group) (other/world)
```

Type	**r**	**w**	**x**	**s**	**t**
File	Read access	Write access	Execute	suid / sgid inherit id	sticky bit
Directory	List files	Add and remove files	Stat / execute files, chdir	New files have dir-gid	Files only deletable by owner

SUID Programs

Each process has *real* and *effective* user / group id

- Usually identical
- Real id
  - Determined by current user
  - `login`, `su`
- Effective ids
  - Determine access rights of a process
  - System calls (e.g., `setuid()`, `setgid()`, etc.)
- `suid/sgid` bits
  - Start process with effective ID different from real ID
  - Attractive targets for attacker

Why does `login` need to be suid root?

No SUID shell scripts anymore

DEMO
suid program
setgid directory

Extended Attributes

```
# lsattr /etc/passwd /etc/ssl
-------------e-- /etc/passwd
----------I--e-- /etc/ssl/certs
```

- Require support from file system
- Management via `lsattr`, `chattr`
  - Undeletable (u)
  - Append only (a)
  - Immutability (i)
  - Secure deletion (s)
  - Compression (c)
  - Hashed trees indexing for directories (I)

DEMO
Extended Attributes

POSIX ACLs

Extend UNIX permission model to support fine-grained access control

```
$ sudo setfacl -m u:pizzaman:r secret
$ getfacl secret
  # file: secret
  # owner: root
  # group: root
  user::rw-
  user:pizzaman:r--
  group::---
  mask::r--
  other::---
```

DEMO

POSIX ACLs

Why use SUID at all?

- ping sends ICMP packets
  - Only privileged programs can access “raw” sockets
  - SUID root solves that problem
- Web server binds to port 80/443
  - Privileged ports (<1024) only root can bind to
  - SUID root would make the entire web-server run as root (what’s the problem?)

Linux Capabilities

- Linux has a fine-grained notion of privilege called *capabilities*
  - Not to be confused with actual capabilities
- Partition root privilege into smaller units
  - `CAP_NET_ADMIN`
  - `CAP_NET_BIND_SERVICE`
  - `CAP_NET_RAW`
  - `CAP_KILL`
  - `CAP_SYS_MODULE`

Linux Capabilities

```
# setcap cap_net_bind_service=+ep /usr/sbin/httpd
# getcap /usr/sbin/httpd
/usr/sbin/httpd = cap_net_bind_service+ep

# find /usr/bin -exec /sbin/getcap {} \;
/usr/bin/gnome-keyring-daemon = cap_ipc_lock+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/systemd-detect-virt =
   cap_dac_override,cap_sys_ptrace+ep
```

Shells

```
# echo $SHELL
/bin/sh
```

- Shells: the classic interface to UNIX systems
  - Interactive REPL environment
  - Also, a convenient programming language
- Program execution, pipelining
  - Fine-grained control of subprocess environment
  - Redirections & pipelining (<, |, and >)
- Many different flavors
  - Bourne shell (sh), Bourne again shell (bash), C shell (csh), Korn shell (ksh)

The Unix Philosophy

Doug McIlroy (1978)

(i) Make each program do ***one thing well***. ...

(ii) Expect the ***output*** of every program to become the ***input*** to another, as yet unknown, program. ...

(iii) ...

(iv) ...

Process System Calls

- **fork** (duplicate current process, create a new process)
- **exec** (replace currently running process with executable)
- **exit** (end process)
- **wait** (wait for a child process)
- **getpid** (get process PID)
- **getpgrp** (get process GID)

Executing Programs

```
int execve(
   const char *path,
   char *const argv[],
   char *const envp[]);
```

- Executing a new program: Invoke the `exec()` syscalls
  - `exec*()` replaces the current program with the program specified as `path`
  - `exec*()` does not return
  - Initializes a new virtual address space
  - Invokes `ld-linux.so.2`, loads shared libs performs runtime linking (ELF, dynamically linked binaries)
  - Invokes interpreter specified in form of #! /path/to/interpreter