

---

# UNIX security

Ulf Larson (modified by Erland Jonsson/Magnus Almgren)  
Computer security group  
Dept. of Computer Science and Engineering  
Chalmers University of Technology, Sweden

# Outline

---

- UNIX security ideas
- Users and groups
- File protection
- Setting temporary privileges
  - Permission bits
  - Program language components
- Examples

# UNIX security ideas

---

- Memory protection for processes
  - Processes have own virtual address space
  - Communication with hardware is done through the operating system
- Files are protected between users
  - The “Everything is a file” concept implies that same mechanisms apply for all objects
- Maintenance is carried out by a “reliable” system administrator
  - Also known as root, or superuser

# Users and groups

---

- A user name is internally represented by a user identifier, or UID
  - Special user names are used for system functions, such as root, guest and apache
  - UNIX *id* command show UID
  - UIDs are stored in file */etc/passwd* with username, preferred shell
  - Your system privileges depends on UID
- Group id, or GID is used to identify *groups* of users
  - UNIX *groups <username>* shows the groups that *<username>* belongs to

# Users and groups (2)

---

- /etc/passwd file entry for user root:
  - root:AAencryptedpw:0:0:root:/root:/bin/bash
- Special user names
  - UNIX comes with special users for administrative purposes: the superuser, or root.
    - As root you can log users out and in, shutdown the computer, start and run network services, run all programs, view all files for all users
    - As root: *most security restrictions are bypassed.*
    - “Hacking root” provides an attacker with unrestricted privileges to a system...**BAD!**

# Users and groups (3)

---

- Sometimes a user need to perform actions as *another* user
  - UNIX *su* command (substitute user / switch user)
    - User enter username and *password* for account. User *becomes* the other user until log out
  - UNIX *sudo* command
    - Run a *single* command usually limited for root (perm. in sudoers file)
    - Users enter their own password (typically) and the access is logged.
  - Executable files with SUID bit set
    - Operating system lets user perform the desired operation with the privileges of the *owner* of the object. When execution finished, user assumes ordinary privileges.
  - Using the `setuid()` function call

# Users and groups (4)

---

- Real and effective UIDs
  - Each user has at any given point in time two (sometimes three) different UIDs
  - Real UID, or RUID is assigned to user when logging in. Used to identify unique user and *remain unchanged*
  - Effective UID, or EUID is initially same as RUID, but changes to *owner* of file during execution of files with the SUID flag set (SUID files).  
EUID changes back to RUID after execution

# File Protection

---

- UNIX file system controls which users can access what items and how
- Simply put: *Everything visible to a user can be represented as a "file"*
  - Each "file" has at least one name, an owner and access rights
  - Running UNIX `/s` command reveals information about files and directories



# File Protection (2): Example

---

```
>>ls -l /home/ulf/example.txt
```

```
-rw-r--r- 1 ulf ulfgrp 1024 Sep 1 11:00 example.txt
```

-	file type
rw-r--r--	file permissions (owner, group, other)
1	# names of the file
ulf	owner
ulfgrp	group
1024	file size
Sep 1 11:00	modification date and time
example.txt	name

# File Protection (3)

---

- File permissions indicate *who* that can do *what* on a specified object.
- 9 characters grouped in 3 *classes* and 3 *kinds* of permissions
- Classes:
  - Owner = The file's owner
  - Group = Users in the file's group
  - Other = Everybody else (except the superuser)
- Kinds:
  - r = Class has *read* access to file,
  - w = Class has *write* access to file,
  - x = Class has *execute* access to file

# File Protection (4)

---

- Example:

- Who can access file a.txt, and in what way:

- `rwX r-- --- usrOne grpTwo a.txt`

Answer:

usrOne has read, write and execute access to a.txt

grpTwo has read access to a.txt

other has *no* access to a.txt

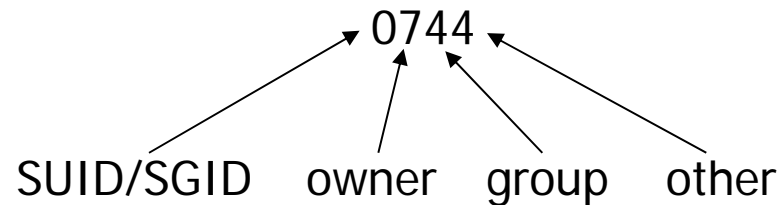
(superuser has full access to a.txt)

# File Protection (5)

- UNIX *chmod* command is used to change file access permissions – 2 different modes

- Octal file permissions

- Four octal numbers are used as follows:



When calculating: r adds 4, w adds 2 and x adds 1 to total.

Example: What is the result (in octal) of setting r,w,x for owner, r for group and r for other for non SUID file?

# File Protection (6)

---

- Combining kinds r, w and x and s with '+', '=' and '-' and classes u, g and o
  - To add write permissions for group: g+w
  - To remove read permissions for other: o-r
  - To set read access for user: u=r
- Example:

Assuming file.txt has permissions 0744, the following two operations achieve the same result

  - >> chmod 0764 file.txt
  - >> chmod g+w file.txt

# Setting temporary permissions: SUID program

---

- A SUID program is a program for which the "s" bit is set
- Used to grant temporary privileges during execution to unprivileged user
  - Example: change the /etc/passwd file
  - What programs are SUID on your system, run  
**find / -perm -4000 -print**
- There are two main methods for changing the s flag through the use of permission bits and chmod
  - SUID bits in file permission.
    - SUID = chmod 4755 file.txt, or chmod u+s file
  - Result: **rws r-x r-x**

# Setting temporary permissions: SUID example

---

- Impact on RUID and EUID from the use of SUID
  - Repeat slide “Users and Groups (4)”
  - During execution of a SUID file, EUID changes to that of the *owner* of the SUID file. The RUID does not change.

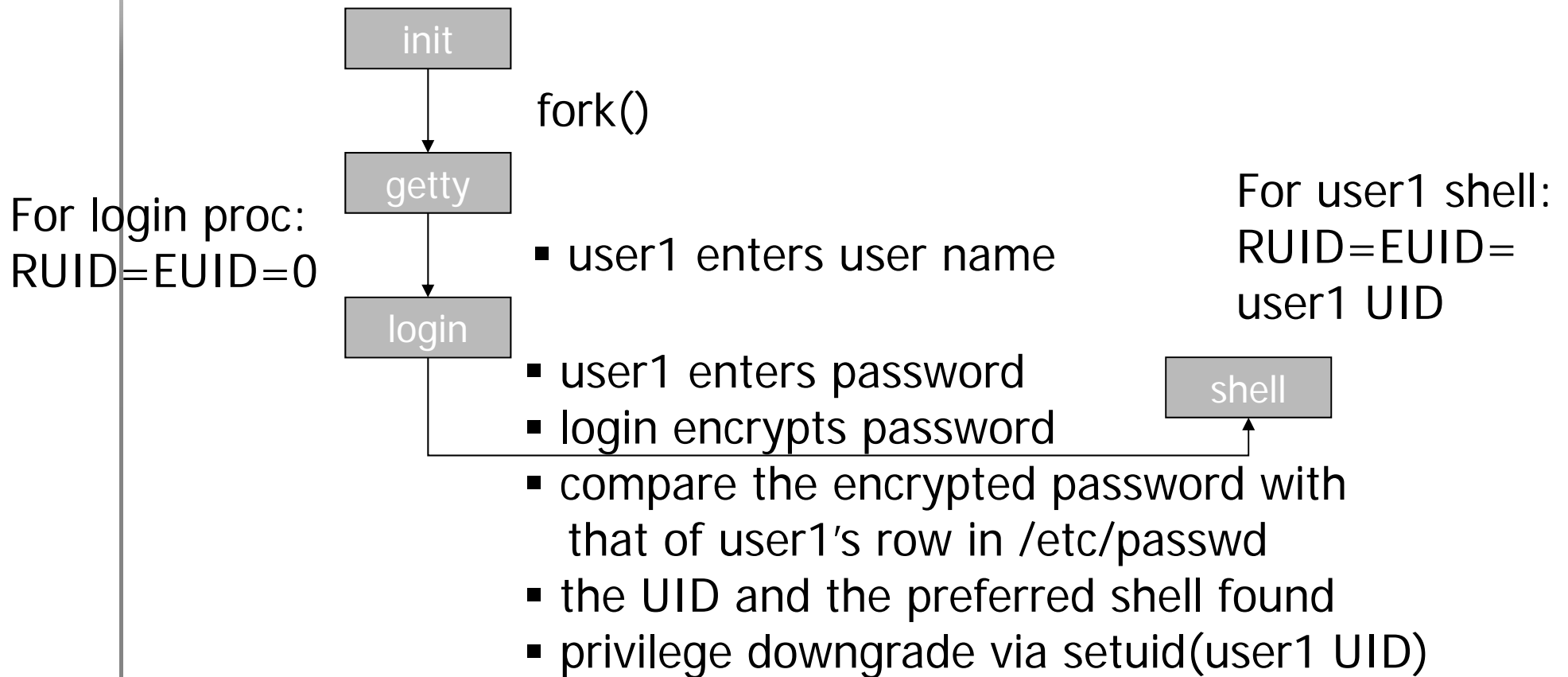
# Setting temporary permissions: `setuid` fcn call

---

- The `setuid()` function call
  - `>>man setuid` (for help – used in Lab 1)
  - Changes the UID of the user to that of the argument of the function call
  - Non-root users can only `setuid` to their own UID.
  - If the caller of the `setuid()` function is non-root, then EUID is changed.
  - If the caller of the `setuid()` function is root, then RUID and EUID is changed. This is used by root to *downgrade* privileges for a user after the user has logged in.



# Example: UNIX login



# Summarizing example 1

---

## ■ Example 1

- User Alice logs in to run the SUID file /home/ulf/becomeMe.exe owned by ulf (UID 12345)

```
>>ls -l becomeMe.exe
```

```
rwsr-xr-x ulf ce becomeMe.exe
```

If user Alice has UID=22448, what are the RUID and EUID before, during and after execution of the file becomeMe.exe?

# Summarizing example (2)

---

## ■ Example 2

– User Alice logs in to run the file

`/home/ulf/dontbecomeMe.exe`

owned by ulf (UID 12345)

```
>>ls -l dontbecomeMe.exe
```

```
rwxr-xr-x ulf ce dontbecomeMe.exe
```

If Alice has the UID 22448, what are the RUID and EUID before, during and after execution of file `dontbecomeMe.exe`?

# Solutions to examples

---

## ■ Example 1

- Before  $RUID=EUID=22448$
- During  $RUID=22448, EUID=12345$
- After  $RUID=EUID=22448$

## ■ Example 2

- Before  $RUID=EUID=22448$
- During  $RUID=EUID=22448$
- After  $RUID=EUID=22448$