LUND UNIVERSITY

EITA25 Computer Security (Datasäkerhet)
Unix (and Linux) Security

PAUL STANKOVSKI WAGNER, EIT, 2020-02-03

Unix (and Linux) Security

- Identification and Authentication
- Access Control
- Other security related things:
  - Devices, mounting file systems
  - Search path
  - TCP wrappers
  - Race conditions
- NOTE: filenames may differ between OS/distributions

Users

- Principals (users) have unique UIDs (user IDs)
  - System cares about ID, not name
  - Several users can have different names but same ID. Then they are treated as the same.
- Superuser (root) has UID = 0
  - There is only one superuser
- Stored in /etc/passwd

- Processes are subjects

UIDs for Processes

- Real user ID – The ID of the logged in principal
  - Can only be changed by root (effective user ID = 0) → this is how login works
- Effective user ID – The ID used for access control
  - Can be changed by root (effective user ID = 0) to anything
    - Used by processes with effective user ID = 0 when they temporarily access files as a less privileged user
  - Can be changed by anyone (any effective user ID) to real user ID
    - This process has to be able to get back to effective user ID = 0

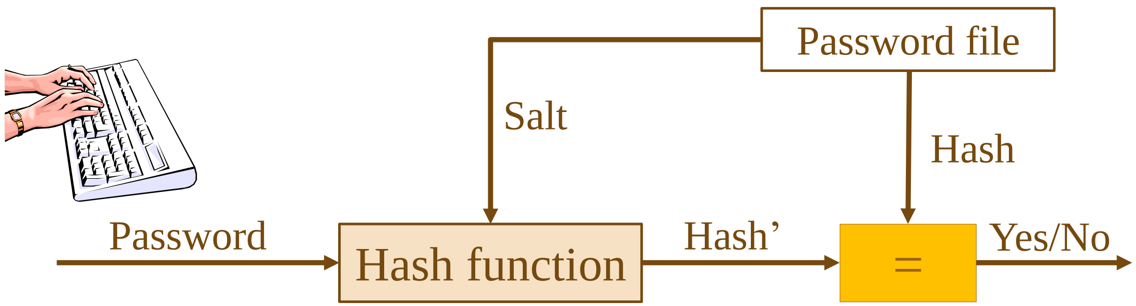
- Same rules apply to group ID

Groups

- Can not associate multiple user IDs with one file
  - We have to put users in groups if we want several users to have access to the file
- Every user belongs to a primary group.
- Older Unix: Can only be in one group at a time
- Newer Unix and Linux: Can be in several groups at the same time
  - New files are associated with current group ID of user
  - Process group ID is the current group ID of user running the process
- Change group (newgrp)
- Primary group given in /etc/passwd
- Secondary groups in /etc/group
  - A group can not belong to a group

```
users:x:100:
Students:x:1000:alice,bob
```

Authentication



- Salt is always used
- Hash function and salt will depend on OS
- We look at three variants

Traditional crypt (Password Hashing)

- Design dates back to 1976
- Based on DES
- Password up to 8 characters, salt 12 bits
  - Take least significant 7 bits → 56 bit key
  - Encrypt zero string 25 times with DES
  - If bit $i = 1$ in salt, swap bits i and $i + 24$ in E-box output
  - Output $12 + 64 = 76$ bits. Encode to 13 characters.

- Problems: Short passwords, short salts, constant cost (and fast function)

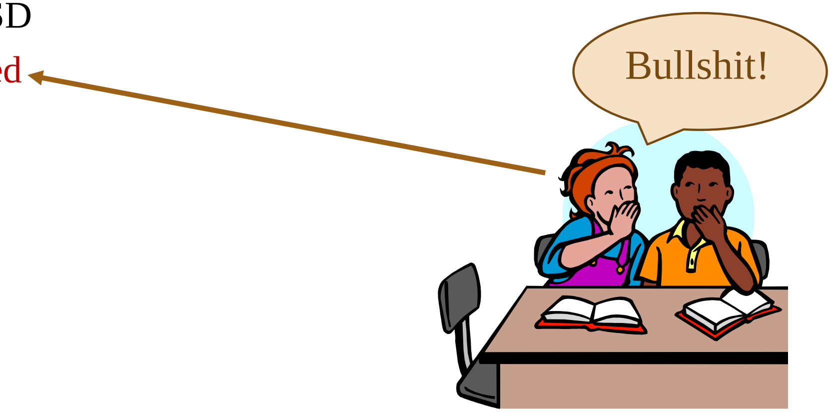
Other Alternatives – MD5 crypt

- MD5 crypt
  - Developed for FreeBSD to avoid export restrictions and allow longer passwords (up to 2^{64} bits)
  - Algorithm uses 1000 iterations → slow
  - Salt 12-48 bits
  - Output: 1 'salt' $ 128 bit hash output

- Problem: Constant cost

Other Alternatives – bcrypt

- Based on block cipher Blowfish
- Password up to 72 characters, 128-bit random salt
- Internal loop with variable cost
- Output $2a$cost$salt + 192 bit hash output
- Default in OpenBSD
- All problems solved



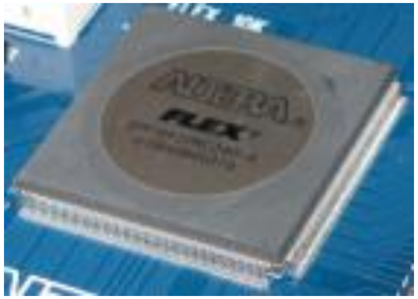
Comparison

	DES crypt	MD5 crypt	bcrypt
Password length	max 8 chars	virtually any	max 72 chars
Salt length	12 bits	12-48 bits	128 bits
Variable cost	No	No	Yes
Evals/sec	1,000,000	10,000	450

- Evals/sec based on 3.2 GHz processor, approximate values given

Final words on our password discussion

- ”All problems solved” is kind of bullshit
- Some devices can be really fast at a low cost
  - With enough money they are really really really fast
  - Several instances can be implemented in parallel
- Can no longer compare
  - CPU – ”needed” when verifying password
  - GPU, FPGA, ASIC – used by attackers



FPGA/ASIC

- Make this more fair by making hashing more difficult (costly) for GPUs, FPGAs and ASICs

GPU

- **Example**: scrypt – requires *memory* as well as CPU cycles

The File /etc/passwd

- Store user (principal) information

Format:

username:password:UID:GID:ID string:home directory:login shell

- File is world readable
- Example:

```
alice:x:1004:100:Alice:/home/alice:/bin/bash
bob:x:1005:100:Bob:/home/bob:/bin/bash
```

The File /etc/shadow

- Save passwords in a non-world readable file
  - Username
  - (Hashed) password
  - Date of last change (days since Jan 1, 1970)
  - Minimum days between password changes (0 means anytime)
  - Maximum days of validity
  - Days in advance to warn user about change
  - Days account is active after password expired
  - Date of account disabling (days since Jan 1, 1970)
  - Last entry is reserved

```
alice:9SuDfhDz3112U:13920:30:180:7:2:14609:
bob:IBDXWbkBirMfU:13920:0:99999:7:::
```

Access Control

- Discretionary access control – owner of file can change permissions
- Three categories: User (owner), Group, Other (world)
- Three access rights: Read, Write, Execute

```
alice@home:>ls -l
totalt 8
drwxr-xr-x 2 alice Students 48 2020-02-03 06:18 directory
-rw-rw-r-- 1 alice Students 22 2020-02-03 06:19 file1
-rw-r--r-- 1 alice Students  9 2020-02-03 06:19 file2
```

Other info from ls -l

Link counter, owner, group, size, date of last change, name

Order of Checking

1. Owner
2. Group
3. Other

Consequence:

if owner = r and other = rw then owner has no write permission

```
alice@home:>ls -l
totalt 0
-r--rw-rw- 1 alice Students 0 2020-02-03 06:20 file
alice@home:>echo hello > file
bash: file: Åtkomst nekas
```

```
bob@home:>ls -l
totalt 0
-r--rw-rw- 1 alice Students 0 2020-02-03 06:20 file
bob@home:>echo hello > file
bob@home:>_
```

Permissions For Directories

- Read = list the directory
- Write = Delete, rename and insert files in directory
- Execute = access directory and access files in directory

```
alice@home:>ls -la
totalt 0
dr-xr-xr-x 2 alice Students  72 2020-02-03 06:21 .
drwxr-xr-x 8 alice Students 384 2020-02-03 06:21 ..
-rw-rw-rw- 1 alice Students   0 2020-02-03 06:21 file
alice@home:>rm file
rm: kan inte ta bort ”file”: Åtkomst nekas
```

```
alice@home:>ls -la
totalt 0
drwxr-xr-x 2 alice Students  72 2020-02-03 06:21 .
drwxr-xr-x 8 alice Students 384 2020-02-03 06:21 ..
-rw-r--r-- 1 root  root       0 2020-02-03 06:21 file
alice@home:>rm -f file
alice@home:>_
```

Change Permissions – `chmod`

- Used to change permissions on files
- Mnemonics can be used: user, group, other, all, read write execute.
- Examples:

```
chmod u+rw file
chmod u=r file
chmod a+rwx file
chmod u-w,g+r,o+r file
chmod a-rwx,u+r file1 file2
```

LUND UNIVERSITY

Change Permissions – `chmod`

- Alternatively, numbers can be used.
- See each group of permissions as one number.
  - Read = 4
  - Write = 2
  - Execute = 1

Sum gives permission

- Example:

```
chmod 754 file
```

```
alice@home:>chmod 754 file; ls -l file
-rwxr-xr-- 1 alice Students 46 2020-02-03 06:22 file
```

Read permission for others
Read and execute for group
Read, write and execute for user

LUND UNIVERSITY

Controlled Invocation

- Some actions require elevated permission
  - Example: Changing password requires root privileges
- Solved by an additional flag
- Allows caller to run program as owner
  - Effective ID of process is ID of program owner (usually root)
  - Users can get general root privileges without root password
- A disadvantage is that this right cannot be given to specified users
  - given to all or group

Setuid and Setgid (programs)

- Effective ID of process is ID of program owner (usually root)
  - Here is the situation when RUID ≠ EUID (real user ID vs. effective user ID)
- Used to temporarily *change* access rights
- *x* is replaced by *s*

```
alice@home:>ls -l
totalt 16
-rwxr-sr-x 1 root root 6378 2020-01-12 15:16 prog_setgid
-rwsr-xr-x 1 root root 6378 2020-01-12 14:58 prog_setuid
alice@home:>./prog_setgid &
[1] 12189
alice@home:>./prog_setuid &
[2] 12190
alice@home:>ps -C prog_setgid,prog_setuid -o pid,ruser,euser,rgroup,egroup,args
  PID RUSER    EUSER    RGROUP   EGROUP   COMMAND
12189 alice    alice    Students root     ./prog_setgid
12190 alice    root     Students Students ./prog_setuid
```

Setuid and Setgid (directories)

- Setuid on directory usually ignored
- Setgid on directory causes new files to get the same group as directory

```
alice@home:>ls -l
totalt 0
drwxr-s--- 2 alice root 48 2020-01-12 15:37 directory
alice@home:>cd directory; touch file; ls -l
totalt 0
-rw-r----- 1 alice root 0 2020-01-12 15:38 file
```

Without setgid, file would get the group which is current group ID for user (set by **`newgrp`** or defaults to primary group).

<u>Allows users to share files more easily</u>

Important SUID Programs

- **`/usr/bin/passwd`** change password
- **`/usr/bin/at`** batch job submission
- **`/bin/su`** change UID program

```
alice@home:>ls -l /usr/bin/passwd /bin/su /usr/bin/at
-rwsr-xr-x 1 root root    31668 2019-04-23 08:48 /bin/su
-rwsr-xr-x 1 root trusted 43940 2019-05-02 09:47 /usr/bin/at
-rwsr-xr-x 1 root shadow  72836 2019-05-02 10:50 /usr/bin/passwd
```

setuid and setgid:

`chmod u+s file` or **`chmod 4XXX file`**

`chmod g+s file` or **`chmod 2XXX file`**

Sticky Bit

- Historically used to keep program code in memory when exiting program
  - still the case in e.g., NetBSD
- Now used to only let owner delete file
  - directory owner and superuser can also delete it

```
bob@home:>ls -la
totalt 0
drwxrwxr-t 2 alice Students 72 2020-02-03 06:52 .
drwxr-x--- 3 alice Students 80 2020-02-03 06:50 ..
-rw-rw-r-- 1 alice Students  0 2020-02-03 06:52 file
bob@home:>rm file
rm: kan inte ta bort ”file”: Operationen inte tillåten
bob@home:>ls -la
totalt 0
drwxrwxr-x 2 alice Students 72 2020-02-03 06:52 .
drwxr-x--- 3 alice Students 80 2020-02-03 06:50 ..
-rw-rw-r-- 1 alice Students  0 2020-02-03 06:52 file
bob@home:>rm file
bob@home:>_
```

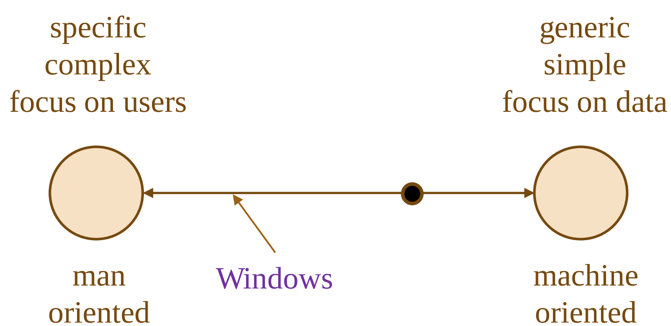
- Typical example: the directory /tmp has sticky bit set

Change Owner and Group (`chown` and `chgrp`)

- **`chown`** is used to change the owner of a file (or directory)
- **`chgrp`** is used to change the group of a file (or directory)
  - **`chown`** can set group also
- Possible problem: A user creates a suid program and owner gets changed to root
- Common solution:
  - Only root can change owner and setuid and setgid bits are removed when owner is changed
  - Anyone can change group to a group they are member of, but setuid and setgid bits are removed when group is changed
- Other solutions possible
  - Let only root use **`chown`**, but preserve setuid and setgid bits
  - Let any user change owner on his/her own files, but remove setuid and setgid bits

Unix Security on the Man-Machine Scale

- Lack of ”flexibility” puts it more to the machine end of the scale
- Limited to read, write and execute
  - E.g., ”shutdown computer” does not exist but may exist in more user-focused environments
  - Can still be implemented though, using the basic access rights



Example: Shutdown in Unix/Linux

- Shutdown can be done with
  - **`/sbin/shutdown`**
  - **`/sbin/halt`**
  - **`/sbin/reboot`**
  - **`/sbin/poweroff`**
- Only root can use these
- Problem: Need to allow some users to shutdown
- Solution (one of several):
  - Add group ”shutdown” in **`/etc/group`**
  - Add users to this group

    `shutdown:x:1500:alice,bob`
  - Use **`chown`** or **`chgrp`** to change group of **`/sbin/shutdown`**

`chown root:shutdown /sbin/shutdown` or **`chgrp shutdown /sbin/shutdown`**

  - Allow group shutdown to execute and set SUID bit since only root is allowed to execute this command

    `chmod u+s,g+x /sbin/shutdown`

The inode

- Stores file information
- Directory contains filename and inode number

```
alice@home:>ls -i
133143 file1  133144 file2  133145 file3  133143 file4
```

Note that file1 and file4 point to the same inode

- inode contains e.g.:
  - Access rights
  - Owner (UID)
  - Group (GID)
  - Time of latest access, modification and change
  - Size of file
  - Pointers to block of data

LUND UNIVERSITY

inode Information (stat)

- Some information about an inode can be found using **`stat`**

```
alice@home:>stat file
  File: ”file”
  Size: 102               Blocks: 8        IO Block: 4096   normal fil
Device: 802h/2050d        Inode: 133060    Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1004/    alice)   Gid: ( 1000/Students)
Access: 2020-02-03 06:55:50.000000000 +0100
Modify: 2020-02-03 06:12:00.000000000 +0100
Change: 2020-02-03 06:55:59.000000000 +0100
alice@home:>_
```

Size in bytes

Inode number

Last access

Access rights given to this file

Last modification of file

Last modification of inode

Default Access Rights (umask)

- Control default permissions, stored in **`/etc/profile`**
- Override in **`~/.profile`** or in prompt
- **`umask`** tells which permissions to exclude by default
- Access = full access AND NOT(umask)
  - Full access for programs and directories: 0777
  - Full access for files: 0666

```
alice@home:>umask 0027; mkdir directory; touch file; ls -l
totalt 0
drwxr-x--- 2 alice Students 48 2020-02-03 06:54 directory
-rw-r----- 1 alice Students  0 2020-02-03 06:54 file
```

Protection of devices

- Devices are treated as files
- Example: If you can read/write physical memory all access control is overruled!
- **`/dev/mem`** is the physical memory
- **`/dev/kmem`** is the virtual memory

LUND UNIVERSITY

Copy files

- Files can be copied in two ways
- **`cp src dest`**
  - Creates a new inode and new physical file owned by user running **`cp`**
- **`ln target linkname`**
  - Creates filename and pointer to target's inode. No new file is created.
  - When one filename is deleted the other is still there and the file is not deleted
  - **`rm`** subtracts the number of links in the inode by 1. If it becomes zero the corresponding data block is freed
- **`ln -s target linkname`**
  - Creates a symbolic link, not a real link
  - When opening symbolic link for reading or writing link is automatically dereferenced
  - If file is deleted, the symbolic link remains, pointing to nothing

Race conditions

- Assume process ”proc” with effective user ID = 0 writes to files in **`/tmp`** directory
  - Process creates, e.g., **`/tmp/file`** and writes temporary data to this file (Proc. opens file for writing and new file is created if it does not exist)
- What if malicious user creates **`/tmp/file`** as symbolic link to **`/etc/passwd`**?
  - The file **`/etc/passwd`** will be overwritten since ”proc” has write access to this file
  - System is damaged
- Race condition: Who creates the file first?

Solutions To This Race Condition

- Create files with unpredictable filenames in **`/tmp`**
  - Still, attacker can try thousands of filenames and will succeed with probability > 0
- Use **`O_EXCL`** flag when opening file
  - Then open fails if file already exists

Function **`mkstemp()`** will do this

- Check if file was opened through a symbolic link
  - Can be done with **`lstat()`**

- All of the above should be used

Mounting File Systems

- Mounting a file system =
  making the particular file system accessible at a specific place in the Linux directory tree

- Different physical devices put under a single root "**/**"
- The mounted file system may contain unwelcome programs
- Options:
  - **nosuid** – turn off SUID and SGID bits
  - **noexec** – no binaries can be executed
  - **nodev** – no devices can be accessed
  - **ro** – read-only
- UIDs and GIDs are local identifiers that may be interpreted differently on different Unix systems
  - Use *global/universally unique* identifiers

Search Path

- When executing programs, system needs to know where to look for them →
  - **PATH** tells system where to look
- **PATH=.:$HOME/bin:/usr/bin:/bin**
  - Programs can be located in current directory + 3 bin directories
  - Trojan horse
- Can be a bad idea to put your current directory in the search path (especially for programs executed by root)
  - At least, put **.** last
  - **PATH=$HOME/bin:/usr/bin:/bin:.**
- Alternatively, call program by its full name

TCP Wrapper

- **`inetd`** is a super-server deamon (starts other servers)
- Config file **`inetd.conf`** maps port numbers to programs

```
ftp      stream  tcp  nowait  root   /usr/sbin/in.ftpd      in.ftpd
telnet   stream  tcp  nowait  root   /usr/sbin/in.telnetd   in.telnetd
```

- Put intermediate program with access control and logging

```
ftp      stream  tcp  nowait  root   /usr/sbin/tcpd         in.ftpd
telnet   stream  tcp  nowait  root   /usr/sbin/tcpd         in.telnetd
```

- The TCP wrapper (**`tcpd`**) will have process name (**`in.ftpd`** and **`in.telnetd`**) and thus know where to go after security checks are done

- **`tcpd`** provides generic network services:
  - Logging (through syslog)
  - Access Control
  - Host Name Verification (client host name spoofing protection)

Network Access Control

- **`/etc/hosts.allow`**: (deamon, client) pair that is allowed access
- **`/etc/hosts.deny`**: (deamon, client) pair that is denied access

Example:

file: **`/etc/hosts.allow`**

```
ALL : localhost
ALL : 192.168.1.2
sshd : ALL EXCEPT .somedomain.com
```

file: **`/etc/hosts.deny`**

```
ALL : ALL
```

Priority:

1. Check **`hosts.allow`**
2. Check **`hosts.deny`**
3. Allow access

Compare with allow/deny in Windows!

LUND
UNIVERSITY