Authorisation

Terminology

- Object – any resource that can be accessed
- Subject – an entity that may try to perform an operation on a resource

- Permission – the right to perform an operation on an object

- Authorisation – managing and enforcing permissions

- Principal – an identity
  - User – a principal that identifies a human
- Authentication – the process of demonstrating that a subject is operating on behalf of a principal

The problem

	Principal1	Principal2	Principal3	…	Principal9999
Object1					
Object2					
Object3					
…					
Object9999					

- Large number of principals(users, clients, etc.)
- Large number of objects (files, computers, other resources)
- Need an efficient way to represent this
  - Efficient both for management and for enforcement

Grouping

- Treat groups of principals or objects uniformly
- Example – a web site has the following groups:
  - Anonymous users
  - Logged-in users
  - Administrators
  - Operators
- Reduces management complexity
- Often reduces enforcement complexity

Example: Linux

- Principals: User and group IDs
- Subjects: Processes
- Each process is associated with a uid and several gids

```
uid=1000(yval) gid=1000(yval)
groups=1000(yval),4(adm),24(cdrom),27(sudo),30(dip),46(plugde
v),108(lpadmin),124(sambashare)
```

Linux File Permissions

- File permissions:

```
-rwxr-xr-x  1 root  wheel  38624 15 Jul 13:59 /bin/ls
```

Linux File Permissions

- File permissions:

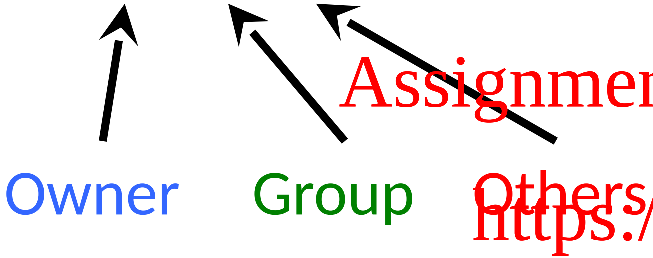
```
-rwxr-xr-x  1 root  wheel  38624 15 Jul 13:59 /bin/ls
```

Owner   Group

Linux File Permissions

- File permissions:

```
-rwxr-xr-x  1 root  wheel  38624 15 Jul 13:59 /bin/ls
```

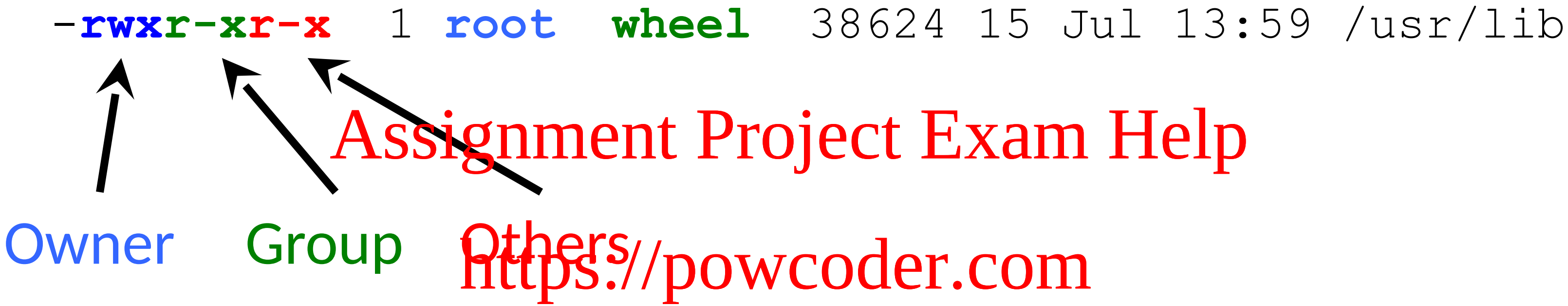


Owner Group Others

- Discretionary Access Control
  - Users can assign permissions to files they own
- Root user (uid 0) can override all permissions

Linux Directory Permissions

- Directory permissions:



```
-rwxr-xr-x  1 root  wheel  38624 15 Jul 13:59 /usr/lib
```

Owner

Group

Others

- What do the permissions mean in the context of directories?
  - R - read – list files in a directory
  - W – write – create, link or unlink files in a directory
  - X – search – use the directory as part of a path

Access Control List

- List of principals and their permissions for each object
- In Linux – extends file permissions:

```
 [root@Maui ~]# getfacl /home/foo/docs/foo.txt
getfacl: Removing leading '/' from absolute path names
# file: home/foo/docs/foo.txt
# owner: jane
# group: executives
user:: r--
user:bob:rw-
user:joe:rwx
group:sales:rwx
group::r--
mask::rwx
other::---
```

Capability-based access control

- Specify the objects and permissions for each subject
- Linux open file descriptors
  - Capabilities for open file access
- Linux Capabilities – allow limited access to the root user. Examples:
  - CAP_DAC_READ_SEARCH – Bypass file read permission checks
  - CAP_DAC_OVERRIDE – Bypass file RWX permission checks
  - CAP_FOWNER – Bypass file ownership checks

Lattice-based access control

- A.k.a label-based access control, rule-based access control
- Associates users and objects with partially ordered labels
- Determine permissions based on the labels
- Military people love it
  - Users with 'confidential' label cannot access 'top-secret' documents

Role-based access control

- Extension of grouping – a *role* is another type of principals
- Subjects assigned to roles
- *At each time a subject has one active role*
- Access rights depend on the active role
- Can be implemented using any of the mechanisms mentioned earlier.
- Example:
  - selinux uses special ACLs for role-based file access

Implementation issues

- Aim for a unified authorisation check

```
int checkperm(subject, object, permission)
```

  - May vary depending on the application!

- Check permission before access
- Do not access if there is no permission

- Consider the level of information to report in case of access denial

TOCTOU

- Authorisation check before every operation may not be enough
  - State may change between check and operation

CVE-2008-2958

```
TMP_DIR=${BASE_TMP_DIR}/`awk 'BEGIN { srand();
     for(i=1;i<22;i++) {
     a=95;
     while (a > 90 && a < 97 {
     a=65+int(50*rand())
     };
     printf("%c", a)
     } }'`
[ -e "$TMP_DIR" ] && rm -rf $TMP_DIR
if [ -e "$TMP_DIR" ]; then
     echo "My temp dir exists already.\nThis looks like a symlink
          attack!"
exit 1
fi
... Some work
mkdir $TMP_DIR
```

CVE-2008-2958

```
TMP_DIR=${BASE_TMP_DIR}/`awk 'BEGIN { srand();
        for(i=1;i<22;i++) {
        a=95;
        while (a > 90 && a < 97 {
        a=65+int(50*rand())
        };
        printf("%c", a)
        } }'`
[ -e "$TMP_DIR" ] && rm -rf $TMP_DIR
if [ -e "$TMP_DIR" ]; then
        echo "My temp dir exists already.\nThis looks like a symlink
                attack!"
exit 1
fi
... Some work
mkdir $TMP_DIR
```

Fix:

```
TMP_DIR=`mktemp -q -d -p "${BASE_TMP_DIR}"
```

Linux network permissions

- Objects: sockets
- Permissions:
  - Create
  - Connect
  - Listen
  - Send/receive data
  - Send raw packets

Linux network permissions

- Objects: sockets
- Permissions:
  - Create: Everyone
  - Connect: Everyone
  - Listen Port < 1024 root, otherwise everyone
  - Send/receive data Everyone
  - Send raw packets Root

- What ports do Web servers listen on? What does that imply?

Linux process permissions

- Permissions:
  - Create
  - Kill
  - Change priority
  - Stop/continue
  - Debug
  - Change resource limits
  - Change security context

Linux process permissions

- Permissions:

- Create	Everyone (security context inherited)
- Kill	User (and root)
- Change priority	Reduce – user, increase – root
- Stop/continue	User
- Debug	User

- Change resource limits
- Change security context

Linux process permissions

- Permissions:

Create	Everyone (security context inherited)
Kill	User (and root)
Change priority	Reduce – user, increase – root
Stop/continue	User
Debug	User
Change resource limits	Soft – user, increase hard – root
Change security context	

Linux process permissions

- Permissions:

Create	Everyone (security context inherited)
Kill	User (and root)
Change priority	Reduce – user, increase – root
Stop/continue	User
Debug	User
Change resource limits	Soft – user, increase hard – root
Change security context	Complex

Changing privileges

- setuid() – sets the uid of the running process
  - Unrestricted use – only root

- Setuid executable – when executed get the uid of the owner

  - Can use setuid() to change between effective and real user ids.

  - Use: sudo – why not login?

- Capabilities – Fine grained privilege escalation

Why capabilities

- Send raw packets Root

- How do we implement ping?
- Setuid binary works, but bugs allow unrestricted access
- Capabilities only allow breaching some network guarantees

Bootstrapping

- First process created with uid=0 (root)
- It creates multiple processes, including a login server
- The login server creates a login process when a user connects
- Login process authenticates the user
- Login process sets the userid to the authenticated user and executes the shell.

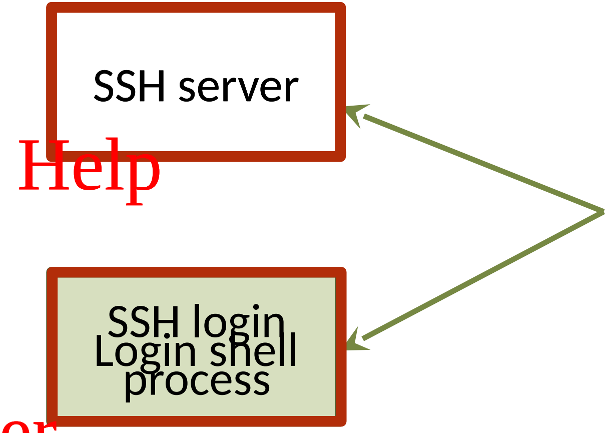
Privilege Separation

Problem description

- Programs often need to run with high privileges
- But at the same time they need to perform a large number of non-privileged operations
- Any bug in the non-privileged code can give elevated privileges to an attacker

Example - SSH

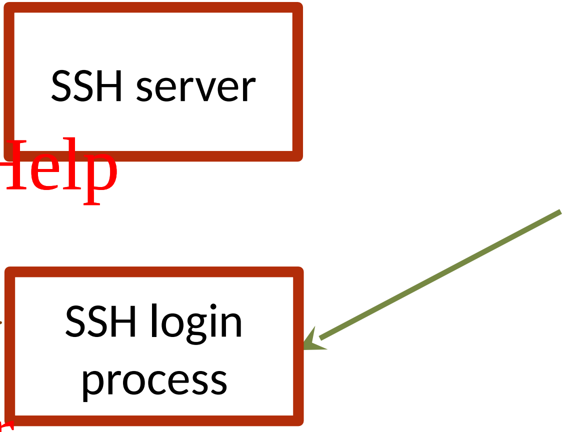
- SSH server – listens for connections
  - Runs as root. Why?
- When the server receives a connection it forks
- … creating a login process
- After authentication the login process drops privileges, becoming the login shell



Example - SSH

- SSH server – listens for connections
  - Runs
- Whe
  conn
- … cre
- After
  process drops privileges, becoming the login shell

Privileged process handling user input



Privilege separation

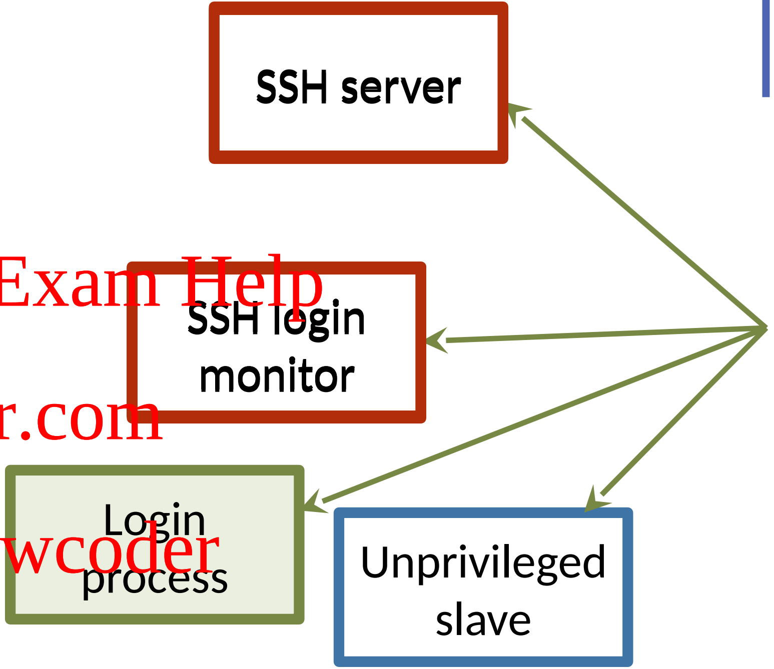
- Break the software into two processes – a privileged monitor and an unprivileged slave

- The slave does the bulk of the work

- The monitor performs privileged operations on behalf of the slave

... In OpenSSH

- SSH Server creates login monitor
- Which creates the unprivileged slave
- The unprivileged slave authenticates the user
- And instructs the monitor to create the login shell



Finer points

- Slave and monitor use IPC mechanisms to communicate
  - Socketpair for all information requests from the slave
  - Shared memory to transfer state from the slave to the login process
- A state machine traces which requests are expected
- The unprivileged slave runs in a chrooted environment

Paper to read

N. Provos, M. Friedl, and P. Honeyman, *Preventing Privilege Escalation*, USENIX Security Symposium, 2003.

http://www.peter.honeyman.org/u/provos/papers/privsep.pdf

Chroot

- chroot("/foo") changes the root of the file system for the process to "/foo"
- Prevents access to files outside "/foo"

- In practice, two changes:

  - The directory name "/" resolves to "/foo"
  - The meaning of the file ".." changes when resolved in "/foo"

- Only the root user can use chroot(). Why?

Linux containers

- Extends chroot environment to create multiple namespaces
- Create a mapping of all of the system identifiers
- Containers have their own file system, user ids, process ids, etc.

Virtualisation

- Virtual machines are abstractions of a computer hardware
- Decouple the operating system from the hardware
- Allows both better isolation and more flexibility than running directly on the hardware
  - The cornerstone of cloud computing

- Options: KVM, Virtual Box, Vmware workstation, Xen, Vmware server, Hyper-V, …