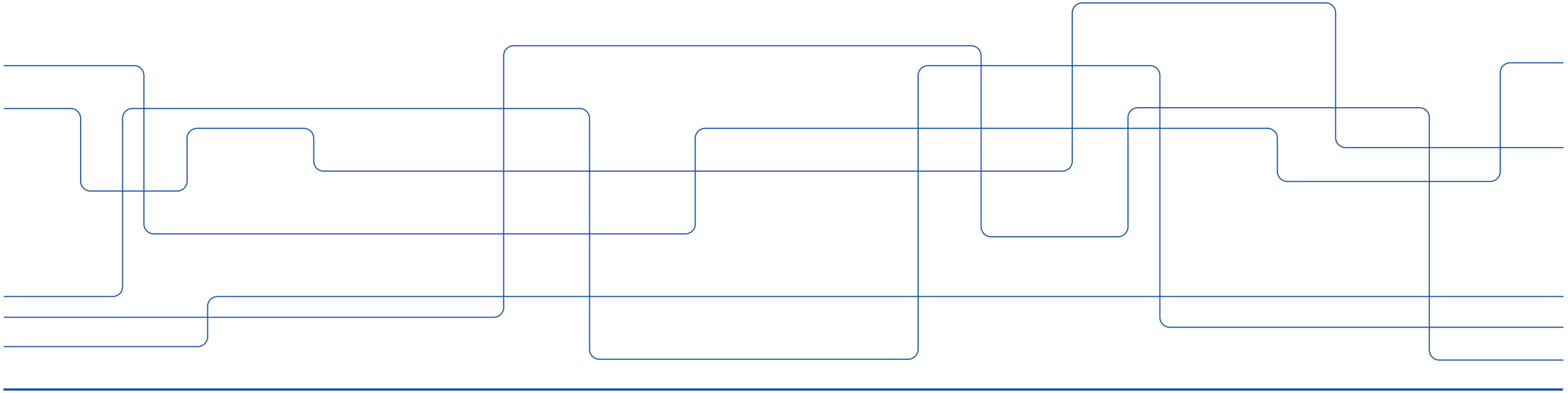
KTH ROYAL INSTITUTE
OF TECHNOLOGY

File System Security

IV1013

Peter Sjödin



File System Security

- Access control
- File access
  - UNIX/Linux and Windows
- Secure storage
  - File encryption
  - Block-level encryption

Access Control

- Users and groups
- Access control lists

- Which users can read/write which files?
- Are my files really safe?
- What does it mean to be root (superuser)?
- What do we really want to control?

Access Control Matrix

- Defines what permissions each subject (user, group, system) has with respect to each object (file, device, resource)
  - > *Collects all access rights in a single place*
  - > *Difficult to manage in practice*

Objects

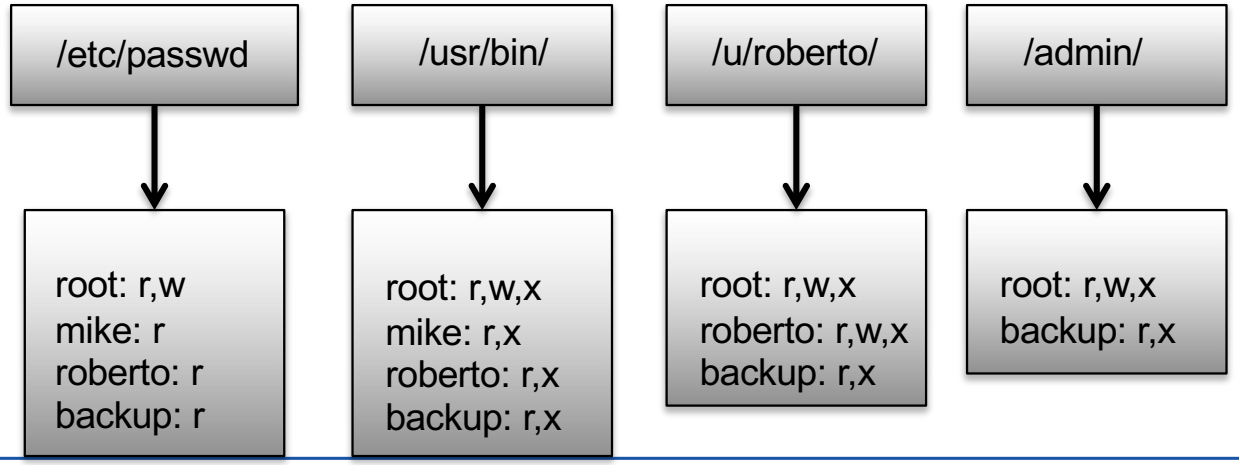
Subjects

	/etc/passwd	/usr/bin/	/u/roberto/	/admin/
root	read, write	read, write, exec	read, write, exec	read, write, exec
mike	read	read, exec		
roberto	read	read, exec	read, write, exec	
backup	read	read, exec	read, exec	read, exec
. . .	. . .	. . .	. . .	. . .

Permissions

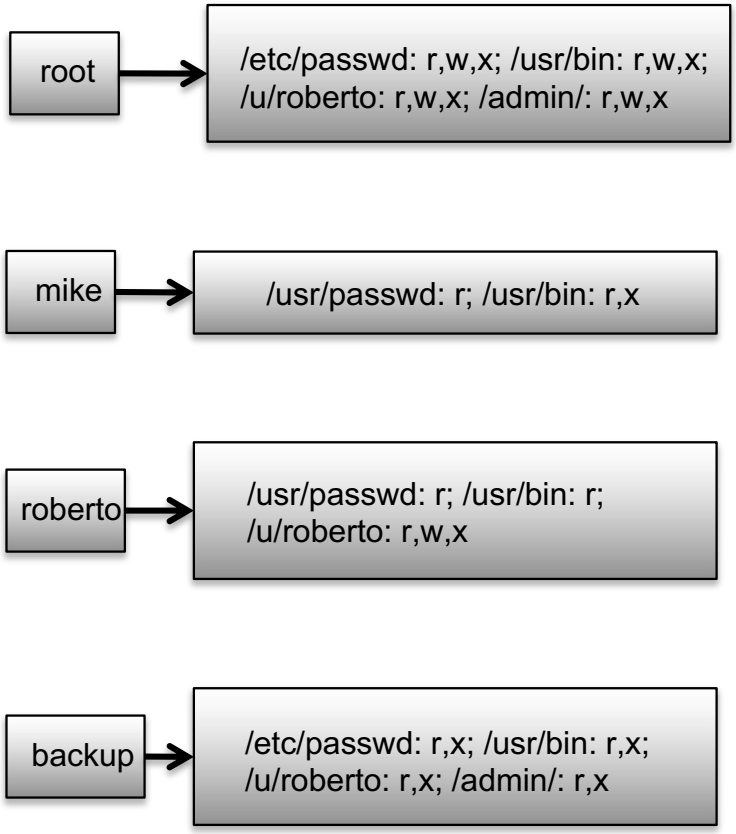
Access Control Lists

- Object-oriented approach
- Each object has a list with permission rights
  - For those subjects that have been granted access
- Can be stored together with the object (file descriptor, for instance)
- Scattered view of system-wide access rights



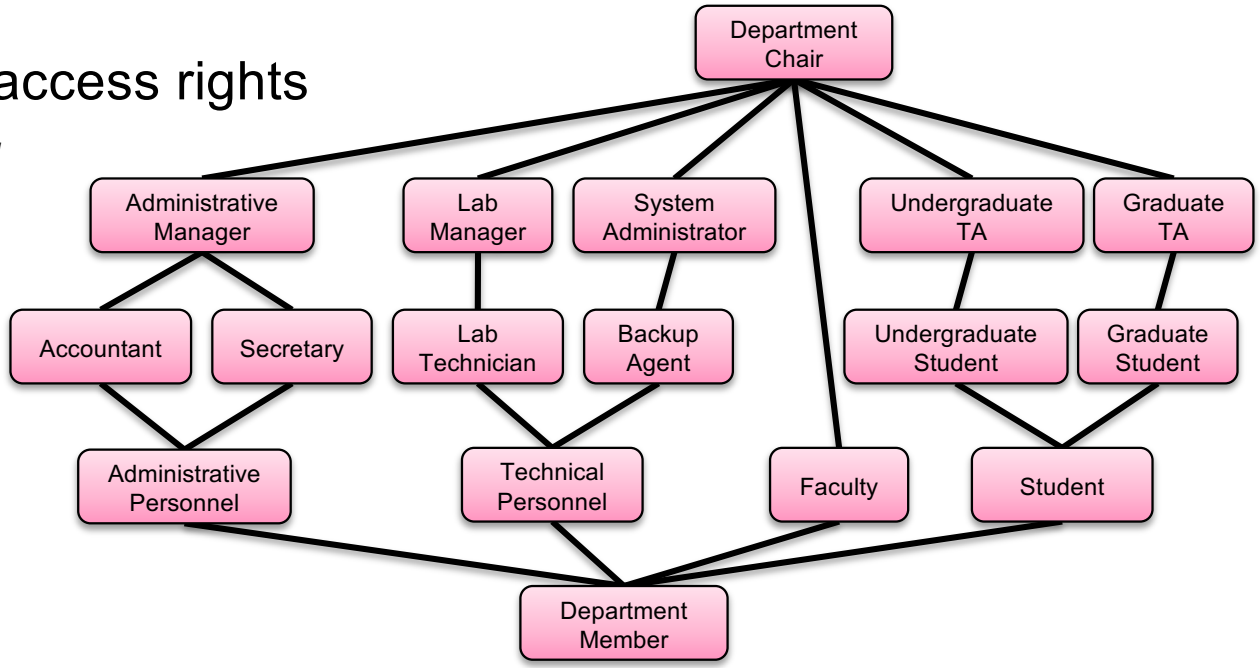
Capabilities

- Subject-centered approach
- Each subject has a list with permission rights
  - For those objects to which the subject has been granted access
- Easy overview of a subject's access rights
- Search list to determine access for a given object



Role-based Access Control

- Define **roles** and then specify access control rights for these roles, rather than for subjects directly
- Combined with hierarchical access rights
  - > *Access rights are inherited*
  - – Keeps number of rules down
- Not often implemented



File System Security

- Access control
- File access
  - UNIX/Linux and Windows
- Secure storage
  - File encryption
  - Block-level encryption

General Principles

- Files and folders (directories) are managed by the operating system
- Applications access files through an API
  - Application Programming Interface
- Access control entry (ACE)
  - Allow/deny a certain type of access to a file/folder by user/group
- Access control list (ACL)
  - Collection of ACEs for a file/folder

- A file handle provides an opaque identifier for a file/folder
- File operations
  - Open file: returns file handle
  - Read/write/execute file
  - Close file: invalidates file handle
- Hierarchical file organization

Closed vs. Open Policy

Closed policy

- Also called “default secure”

- Give Tom read access to “foo”
- Give Bob read/write access to “bar”
- Tom: I would like to read “foo”
  - > *Access allowed*
- Tom: I would like to read “bar”
  - > *Access denied*

Open Policy

- Deny Tom read access to “foo”
- Deny Bob read/write access to “bar”
- Tom: I would like to read “foo”
  - > *Access denied*
- Tom: I would like to read “bar”
  - > *Access allowed*

Linux File Access

- Closed policy
  - Allow-only ACEs
- Access to file depends on ACL of file and of all its ancestor folders
  - Start at root of file system
  - Traverse path of folders
  - Each folder must have “execute” permission
    - *“cd” – change directory*
- Different paths to same file not equivalent
- File’s ACL must allow requested access

Linux File System Internals

- A file is represented internally by an inode data structure
  - > *Try "ls -i"*
- Each inode is uniquely identified by its number (not name!)
  - A directory is a file that maps names to inode numbers
  - The same inode can be referenced in several directories
    - > *"hard links"; create with "ln" command*
    - > *Reference counter in inode*
  - Unlink ("rm"): decrease reference count
    - > *File deleted when reference counter goes to zero*

/users/rufus/	
name	*inode*
.login	39789
keydata.txt	81341
datafiles	47422

/users/rufus/datafiles/	
name	*inode*
data1.txt	98112
data2.txt	81341

Inode 81341	
Refcount (= 2)	*Protection*
Creation time	…
File data	

Linux Symbolic Links

- "Soft links" or "symlinks"
  - To a target file or directory
  - Create with "ln –s"
  - Special kind of entry in directory file:
    - > *Instead of storing inode number, the path to the target is stored*
    - > *View target's path through "ls -l"*
  - The same file or directory can have multiple symlinks to it
  - Removal of symlink does not affect target
    - > *Removal of target invalidates (but not removes) symlinks to it*
      - Leaves "stale" symlinks
  - Analogue of Windows "shortcut" or Mac OS "alias"

Linux File Access Protection

- File protection depends on the protection of the file (inode) itself, as well on the protection of all ancestor directories
- Hence, file protection may vary depending on where in the file system the inode is referenced

Windows File Access

- "Closed Policy with Negative Authorization and Deny Priority"
  - > *Allow and deny ACEs*
  - > *By default, deny ACEs precede allow ACEs*
- Access to file depends only on file's ACL
  - > *ACLs of ancestors ignored when access is requested*
  - > *Permissions set on a folder usually propagated to descendants (inheritance)*
    - System keeps track of inherited ACE's

- Give Tom read/write access to "bar"
- Deny Tom write access to "bar"
- Tom: I would like to read "bar"
  - Access allowed
- Tom: I would like to write "bar"
  - Access denied

Windows File Access (2)

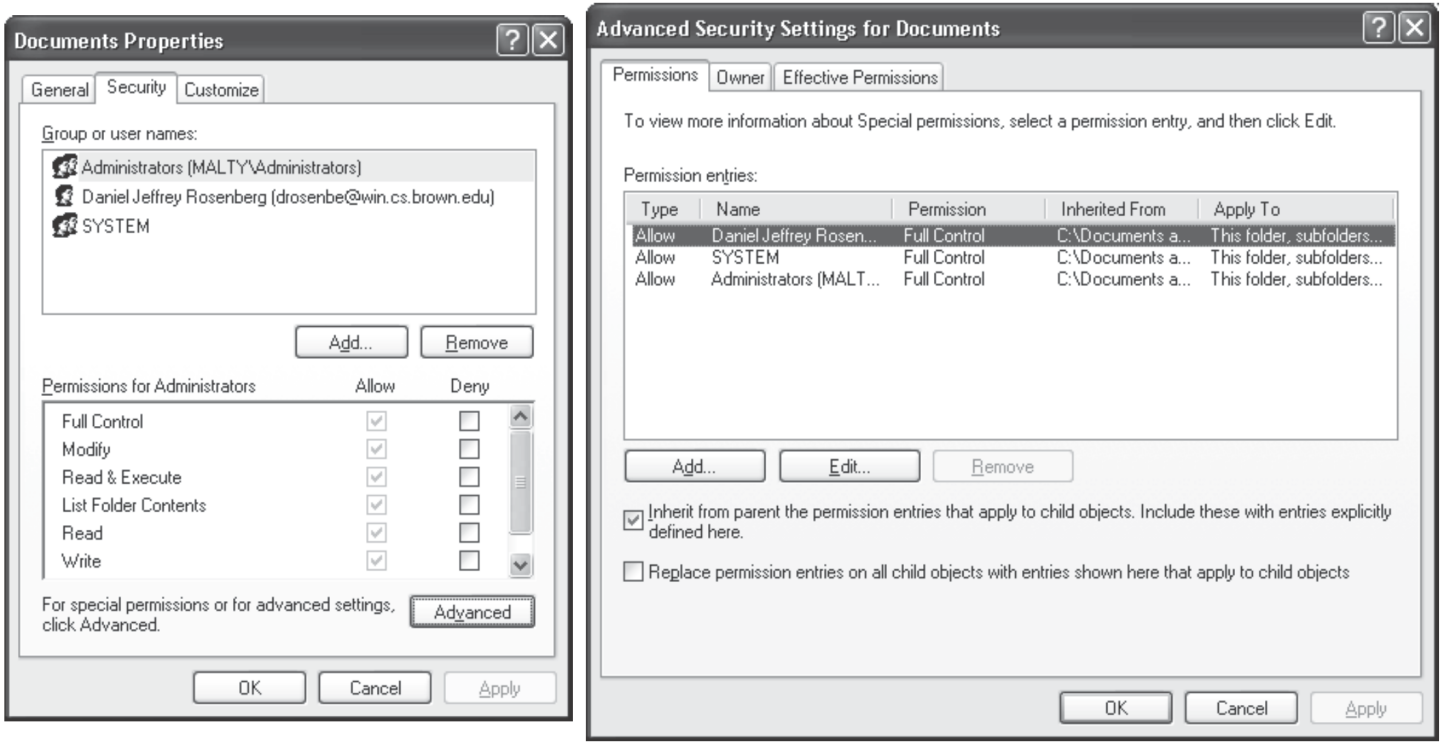
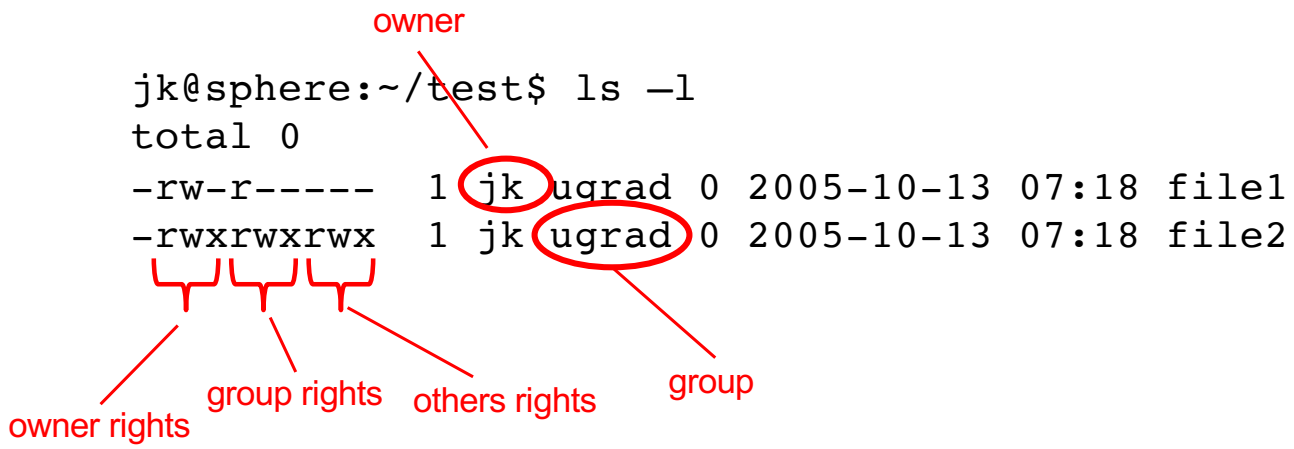


Figure 3.13: Customizing file permissions in Windows XP.

Unix Permissions

- Standard for all UNIXes
- Every file is owned by a user and has an associated group
- Permissions often displayed in compact 10-character notation

> *ls –l*



Permissions for Directories

- Permissions bits interpreted differently for directories
- Read bit allows listing names of files in directory, but not their properties like size and permissions
- Write bit allows creating and deleting files within directory
- Execute bit allows entering the directory and getting properties of files in the directory
- Permissions in "`ls –l`" output begin with "d":

```
jk@sphere:~/test$ ls –l
Total 4
drwxr-xr-x  2 jk ugrad 4096 2005-10-13 07:37 dir1
-rw-r--r--  1 jk ugrad    0 2005-10-13 07:18 file1
```

Permissions Examples (Directories)

`drwxr-xr-x`	all can enter and list the directory, only owner can add/delete files
`drwxrwx---`	full access to owner and group, forbidden to others
`drwx--x---`	full access to owner, group can access known filenames in directory, forbidden to others
`drwxrwxrwx`	full access to everyone

Setuid/setgid bits

- Set-user-ID (“suid” or “setuid”) bit
  - On executable files, causes the program to run as file owner regardless of who runs it
- Set-group-ID (“sgid” or “setgid”) bit
  - Corresponding for the file’s group
- A way to temporarily elevate user access rights for specific purpose
  - Updating system files, for instance
  - Example: ”passwd” program to change a user’s password
    - > *Update /etc/shadow – read/write protected for all users*

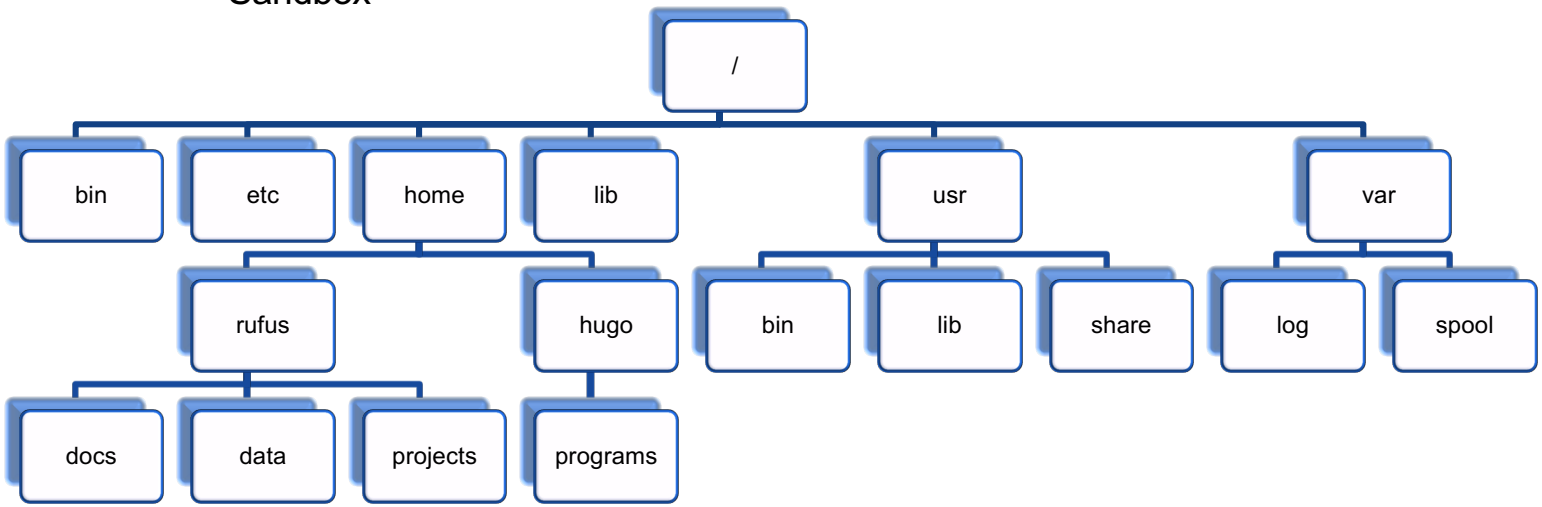
```
jk@sphere:~/test$ ls –l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 Jan 27 01:50 /usr/bin/passwd
```

Setuid Programs

- Users can execute commands with root privileges
  - > *Without logging in as root with “su” or “sudo”*
- Security concerns
  - > *Attacker manipulates setuid program*
    - Buffer overflow, incomplete input validation, …
  - > *May be able to execute arbitrary code with system privileges*

File System Root

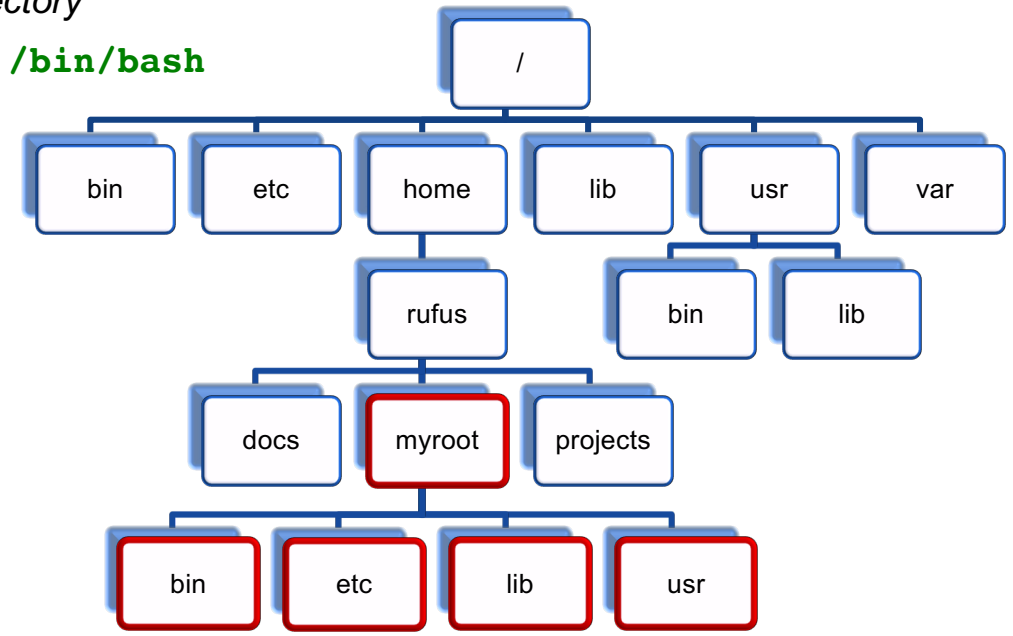
- Programs execute in the context of a hierarchical file system
  - > *With a well-defined root (inode with top-level directory)*
  - – Programs can be executed with an alternative root
    - > *Limit the context of the program to a controlled environment*
      - “Sandbox”



Chroot Command

- Limit the context of the program to a different root directory
  - "Sandbox"
- "**`chroot /home/rufus/myroot /bin/bash -i`**"
  - > *Run bash with **`/home/rufus/myroot`** as root directory*
    - Will run **`/home/rufus/myroot/bin/bash`**, not **`/bin/bash`**
  - > *Can't reach outside **`/home/rufus/myroot`***

Container-based virtualization
(as in LXC and Docker) is "glorified chroot"



POSIX ACLs

- Limitations with regular UNIX/Linux ACLs
  - Three levels only: owner, group, and others
- Extended ACLs
  - Allows access rights to be defined for each user and group
  - Flexible file ACLs
  - POSIX ACL extension
  - POSIX – Portable Operating System Interface
    - > *IEEE Standards for operating systems compatibility*
      - Linux is “mostly POSIX compliant”
- Combined with regular UNIX/Linux file protection

Extended ACL Example

- Give user “student” read access
- All other users cannot read file
  – unless owner or group

```
rufus@ubuntu:~ $ ls –l
total 42
-rw-rw---- 1 rufus rufus 42 May  2 10:44 file.txt
rufus@ubuntu:~ $ setfacl -m u:student:r file.txt
rufus@ubuntu:~ $ ls –l
total 42
-rw-rw----+ 1 rufus rufus 42 May  2 10:44
rufus@ubuntu:~ $ getfacl file.txt
# file: file.txt
# owner: rufus
# group: rufus
user::rw-
user:student:r–
group::rw-
mask::rw-
other::---
```

Questions

Role-based access control has several advantages, but many questions arise when it comes to practical implementation. How can inheritance be realized? How can parallel units (labs, departments, …) be dealt with? Can a subject have roles at different levels/units? Discuss!

Why do we say that the UNIX/Linux file system is organized as a Directed Acyclic Graph, and not a tree?
In UNIX/Linux, hard links are not allowed for directories. Why?

Consider an access matrix for an operating system. What do you think is a realistic proportion of number of rows (subjects) versus columns (objects)? What are the implications for ACL-based versus capability-based access control?

Windows has both “allow” and “deny” ACEs. Why? Wouldn’t it be sufficient with only one of them, plus a suitable default policy?

File System Security

- Access control
- File access
  - UNIX/Linux and Windows
- Secure storage
  - File encryption
  - Block-level encryption

File and Disk Encryption

- Disk encryption
  - Encryption of physical or logical drive
  - Block-level encryption
    - *BitLocker in Windows*
    - *LUKS in Linux (Linux Unified Key Store)*
    - *FileVault in Mac OS X*
- File system encryption
  - Encrypt files or directories
    - *Encrypting File System in Windows*
    - *eCryptfs in Linux*
    - *DiskUtil in Mac OS X*
      - Encrypted disk image
- Third-party applications
  - Commercial and open source

Lost Laptops

- Study by Ponemon Institute in 2009
  - > *Average cost of a lost laptop for a corporation is $50K*
  - > *Costs include data breach, intellectual property loss, forensics, legal and regulatory expenses*
  - – Data breach much more serious than hardware loss
    - > *Encryption decreases cost by $20K*
  - – The sooner the loss is discovered, the lower the cost
    - $8,950 if discovered same day, $115,849 if it takes more than a week
  - – Loss of productivity is minor problem (1% of total cost)
- Data can also be copied while laptop is unattended

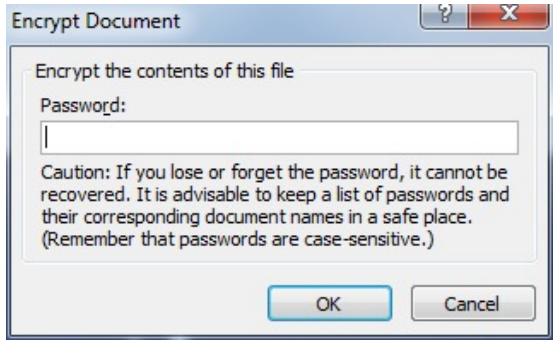
The Cost of a Lost Laptop, Study by Ponemon Institute (sponsored by Intel Corp.), 2009.
http://www.intel.se/content/dam/doc/white-paper/enterprise-security-the-cost-of-a-lost-laptop-paper.pdf

Other Data Protection Scenarios

- Defending against loss of USB drives and smartphones
- Defending against data-stealing malware
- Defending against equipment seizure
- Donating decommissioned machines
- Recycling obsolete or faulty machines
- Off-site backups
- Cloud storage

Sharing Encrypted Files

- Solution A
  - Encrypt file with symmetric key K
  - Share K with authorized users
  - Users need to keep many keys
  - User revocation requires redistributing new key
- Solution B
  - Different symmetric keys K_1, …, K_n for authorized users
  - Encrypt file multiple times with K_1, …, K_n
  - Inefficient in terms of space and computing time
- Solution C
  - Encrypt file with single symmetric key K
  - Encrypt K with public keys of authorized users PK_1, …, PK_n
  - Store with file $E_{PK1}(K)$, …, $E_{PKn}(K)$

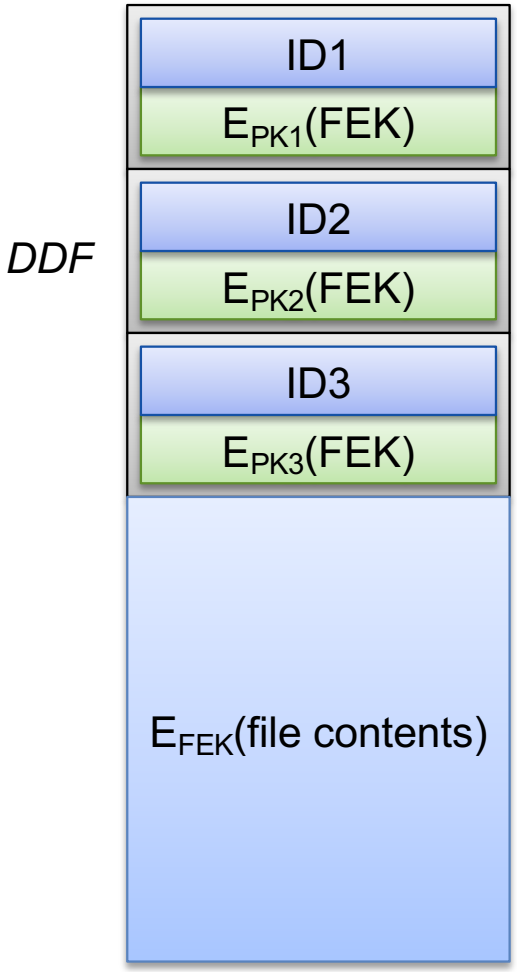


Encrypting File System (EFS)

- Available in Windows since Windows 2000
  - Per-file encryption
    - > *Protects file content but not file name and other metadata*
  - Supports sharing of encrypted files
    - > *Keys unlocked on successful user login*
  - Latest version uses RSA, SHA-256, and AES

EFS Keys

- Users have public-private key pairs
- Each file is encrypted with a different symmetric File Encryption Key (FEK)
- FEK is encrypted with public key of file owner (and other authorized users)
- Data Decryption Fields (DDF) stored in file header (metadata)
  - ID of authorized user
  - FEK encrypted with public key of user



EFS Issues

- Protection only local to file system
  - File copied to another file system is decrypted
  - Email attachment sent decrypted
  - File content may be leaked to unprotected temporary files
- Removing authorized user
  - DDF of revoked user removed from file header
  - File should be re-encrypted with new FEK, but is not …

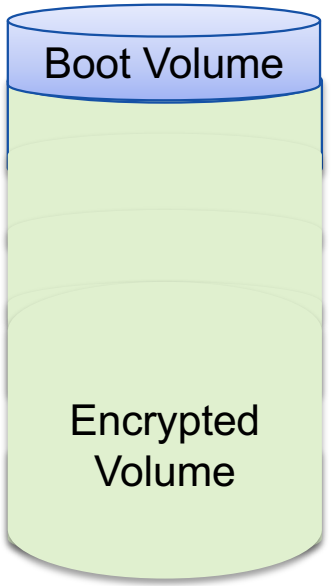
BitLocker

- Block-level encryption for Windows
- Targets lost-laptop scenario
- Encrypts NTFS volumes
- All disk sectors encrypted with symmetric encryption method
- Key can be provided by user at boot time
  - Passphrase
  - Hardware token
- Key can be stored in special cryptographic chip that releases it to the operating system only after verifying the integrity of the system
  - Trusted Platform Module (TPM)

Bruce Schneier's blog post on "Encrypting Windows Hard Drives", June 15, 2015
https://www.schneier.com/blog/archives/2015/06/encrypting_wind.html

BitLocker Architecture

- Two volumes
  - > *Small unencrypted boot volume*
  - > *Large encrypted volume storing rest of OS and user files*
- Two keys
  - > *Volume Master Key (VMK)*
    - Unlocked through authentication procedure
  - > *Full Volume Encryption Key*
    - Used to encrypt sectors of encrypted volume
    - Stored on boot volume encrypted with VMK
    - Kept in memory and never written unencrypted to disk
- For each disk sector accessed
  - > *Decrypt on read*
  - > *Encrypt on write*



TrueCrypt

- Free open-source disk encryption software for Windows, Mac OS X, and Linux
- Creates an encrypted area (virtual encrypted disk) inside an ordinary file
  - > *Disk image*
- When the user provides the correct password, the file is mounted and becomes a volume
  - > *Just like inserting a USB drive*
- Files copied to/from this volume are encrypted/decrypted on the fly, automatically and transparently

TrueCrypt abolished in 2014 – “For reasons that remain a titillating source of hypothesis, intrigue and paranoia”
https://www.grc.com/misc/truecrypt/truecrypt.htm

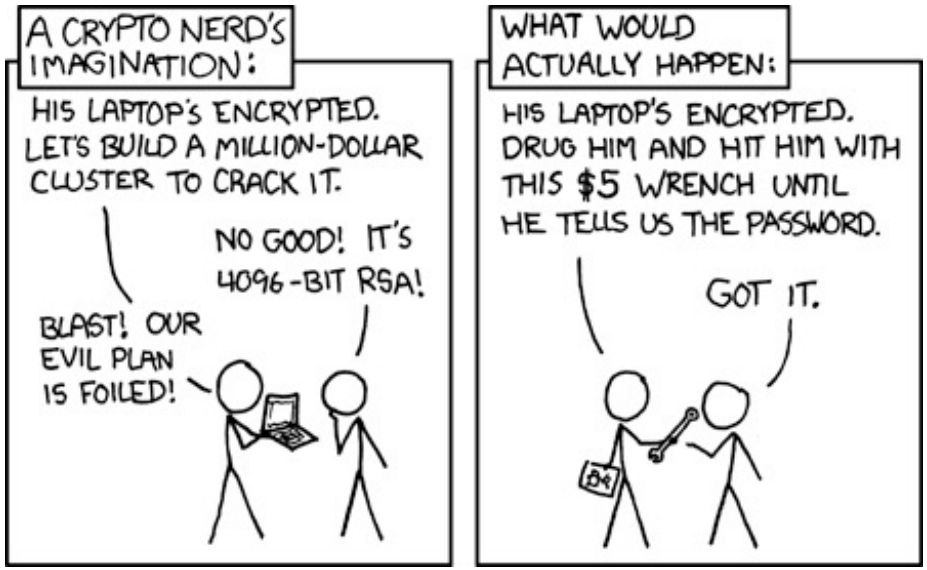
Plausible Deniability

  - If illegal operations are discovered, it should be possible to deny any connection or guilt of the principals

- In general, plausible deniability refers to
  - > *Any act that leaves little or no evidence of irregularities or abuse*
  - > *In computer parlance, it is the ability to deny the presence of data hidden within a container*

- Alice is a human-rights worker with sensitive information on her laptop
- She is concerned that the secret police will seize her computer and ask her to reveal the decryption key
- She needs to protect her data in such a way that her encrypted files are deniable:
  - Nothing should reveal to the secret police that there are hidden files on her computer

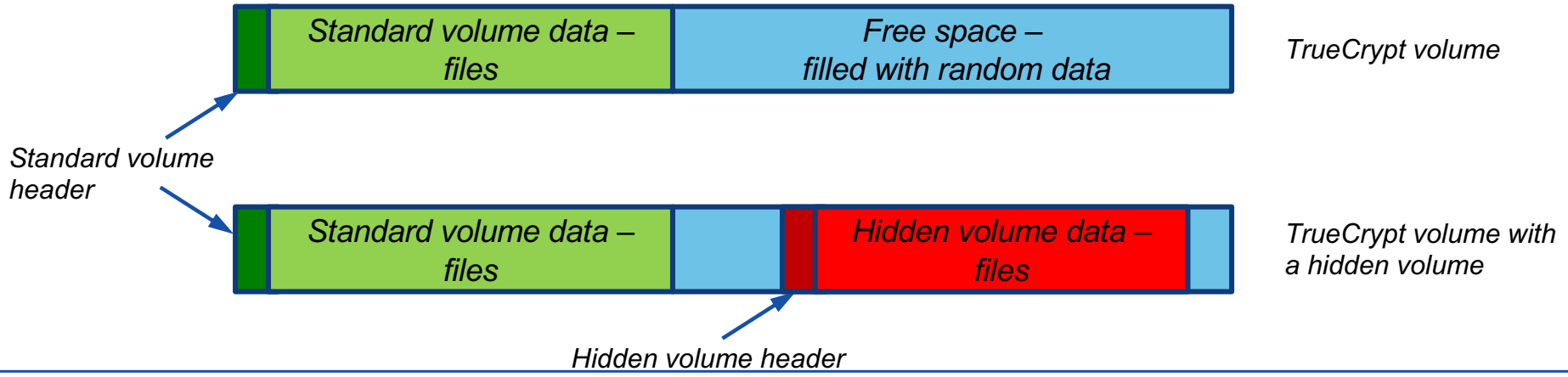
Xkcd Comic on Security



http://xkcd.com/538/

TrueCrypt Hidden Volume

- A TrueCrypt volume placed in the free space of another TrueCrypt volume
  - No information in the header that there is a hidden volume
  - Decrypted by the user asking TrueCrypt to search for a hidden volume, with a given password



Assignment Hidden Encryption

- Your task here is to store encrypted data in the file system
  - > *Data in a binary file*
  - > *Unused part of a disk partition*
  - > *…*
  - – Write programs that can store and recover encrypted data
    - > *Encryption with AES in CTR mode*

File System Security

- Access control
- File access
  - UNIX/Linux and Windows
- Secure storage
  - File encryption
  - Block-level encryption