Filesystem Security

General Principles

- Files and folders are managed by the operating system
- Applications, including shells, access files through an API
- 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
  - Tree (Windows)
  - DAG (Linux)

Discretionary Access Control (DAC)

- Users can protect what they own
  - The owner may grant access to others
  - The owner may define the type of access (read/write/execute) given to others
- DAC is the standard model used in operating systems
- Mandatory Access Control (MAC)
  - Alternative model not covered in this lecture
  - Multiple levels of security for users and documents
  - Read down and write up principles

Closed vs. Open Policy

Closed policy

- Also called “default secure”
- Give Tom read access to “foo”
- Give Bob r/w 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 r/w access to “bar”
- Tom: I would like to read “foo”
  - Access denied
- Tom: I would like to read “bar”
  - Access allowed

Closed Policy with Negative Authorizations and Deny Priority

- Give Tom r/w 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
- Policy is used by Windows to manage access control to the file system

Access Control Entries and Lists

- An Access Control List (ACL) for a resource (e.g., a file or folder) is a sorted list of zero or more Access Control Entries (ACEs)
- An ACE refers specifies that a certain set of accesses (e.g., read, execute and write) to the resources is allowed or denied for a user or group
- Examples of ACEs for folder “Bob’s CS167 Grades”
  - Bob; Read; Allow
  - TAs; Read; Allow
  - TWD; Read, Write; Allow
  - Bob; Write; Deny
  - TAs; Write; Allow

Linux vs. Windows

- Linux
  - 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 (cd) permission
  - Different paths to same file not equivalent
  - File's ACL must allow requested access
- Windows
  - Allow and deny ACEs
  - By default, deny ACEs precede allow ones
  - 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

Linux File Access Control

- File Access Control for:
  - Files
  - Directories
  - Therefore…
    - \dev\ : *devices*
    - \mnt\ : *mounted file systems*
    - What else? *Sockets, pipes, symbolic links…*

Linux File System

- Tree of directories (folders)
- Each directory has links to zero or more files or directories
- Hard link
  - From a directory to a file
  - The same file can have hard links from multiple directories, each with its own filename, but all sharing owner, group, and permissions
  - File deleted when no more hard links to it
- Symbolic link (symlink)
  - From a directory to a target file or directory
  - Stores path to target, which is traversed for each access
  - 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
  - Analogue of Windows shortcut or Mac OS alias

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
- To see permissions, use `ls -l`

```
jk@sphere:~/test$ ls -l
total 0
-rw-r-----  1 jk ugrad 0 2005-10-13 07:18 file1
-rwxrwxrwx  1 jk ugrad 0 2005-10-13 07:18 file2
```

Permissions Examples (Regular Files)

-rw-r—r--	read/write for owner, read-only for everyone else
-rw-r-----	read/write for owner, read-only for group, forbidden to others
-rwx------	read/write/execute for owner, forbidden to everyone else
-r--r--r--	read-only to everyone, including owner
-rwxrwxrwx	read/write/execute to everyone

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 the directory
- *Execute* bit allows entering the directory and getting properties of files in the directory
- Lines for directories in `ls -l` output begin with d, as below:

```
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)

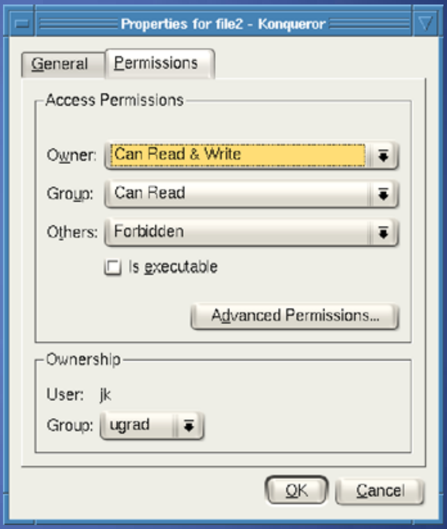
drwxr-xr-x	all can enter and list the directory, only owner can add/delete files
drwxrwx---	full access to owner and group, forbidden to others
drwx--x---	full access to owner, group can access known filenames in directory, forbidden to others
-rwxrwxrwx	full access to everyone

File Sharing Challenge

- Creating and modifying groups requires root
- Given a directory with permissions drwx------x and a file in it
  - Give permission to write the file to user1, user2, user3, … without creating a new group
  - Selectively revoke a user
- Solution 1
  - Give file write permission for everyone
  - Create different random hard links: user1-23421, user2-56784, …
- Problem! Selectively removing access: hard link can be copied
- Solution 2
  - Create random symbolic links
- Problem! Symbolic link tells where it points

Working Graphically with Permissions

- Several Linux GUIs exist for displaying and changing permissions
- In KDE's file manager Konqueror, right-click on a file and choose Properties, and click on the Permissions tab:
- Changes can be made here (more about changes later)



Owner: Can Read & Write
Group: Can Read
Others: Forbidden
☐ Is executable
User: jk
Group: ugrad

Special Permission Bits

- Three other permission bits exist
  - Set-user-ID ("suid" or "setuid") bit
  - Set-group-ID ("sgid" or "setgid") bit
  - Sticky bit

Set-user-ID

- Set-user-ID (“suid” or “setuid”) bit
  - On executable files, causes the program to run as file owner regardless of who runs it
  - Ignored for everything else
  - In 10-character display, replaces the 4th character (x or -) with s (or S if not also executable)
    - -rwsr-xr-x: setuid, executable by all
    - -rwxr-xr-x: executable by all, but not setuid
    - -rwSr--r--: setuid, but not executable - not useful

Set-group-ID

- Set-group-ID (“sgid” or “setgid”) bit
  - On executable files, causes the program to run with the file’s group, regardless of whether the user who runs it is in that group
  - On directories, causes files created within the directory to have the same group as the directory, useful for directories shared by multiple users with different default groups
  - Ignored for everything else
  - In 10-character display, replaces 7th character (x or -) with s (or S if not also executable)
    - -rwxr-sr-x: setgid file, executable by all
    - drwxrwsr-x: setgid directory; files within will have group of directory
    - -rw-r-Sr--: setgid file, but not executable - not useful

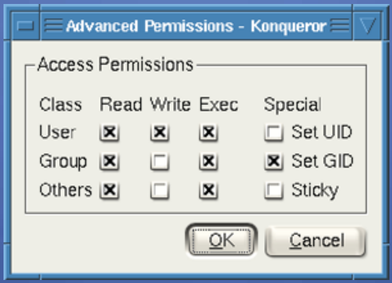
Sticky Bit

- On directories, prevents users from deleting or renaming files they do not own
- Ignored for everything else
- In 10-character display, replaces 10th character (x or -) with t (or T if not also executable)

drwxrwxrwt: sticky bit set, full access for everyone
drwxrwx--T: sticky bit set, full access by user/group
drwxr--r-T: sticky, full owner access, others can read *(useless)*

Working Graphically with Special Bits

- Special permission bits can also be displayed and changed through a GUI
- In Konqueror's Permissions window, click Advanced Permissions:
- Changes can be made here (more about changes later)


Advanced Permissions - Konqueror

Access Permissions

Class	Read	Write	Exec	Special
User	☒	☒	☒	☐ Set UID
Group	☒	☐	☒	☒ Set GID
Others	☒	☐	☒	☐ Sticky

Root

- “root” account is a super-user account, like Administrator on Windows
- Multiple roots possible
- File permissions do not restrict root
- This is *dangerous*, but necessary, and OK with good practices

Becoming Root

- su
  - Changes home directory, PATH, and shell to that of root, but doesn’t touch most of environment and doesn’t run login scripts
- su -
  - Logs in as root just as if root had done so normally
- sudo <command>
  - Run just one command as root
- su [-] <user>
  - Become another non-root user
  - Root does not require to enter password

Changing Permissions

- Permissions are changed with chmod or through a GUI like Konqueror
- Only the file owner or root can change permissions
- If a user owns a file, the user can use chgrp to set its group to any group of which the user is a member
- root can change file ownership with chown (and can optionally change group in the same command)
- chown, chmod, and chgrp can take the -R option to recur through subdirectories

Examples of Changing Permissions

chown -R root dir1	Changes ownership of dir1 and everything within it to root
chmod g+w,o-rwx file1 file2	Adds group write permission to file1 and file2, denying all access to others
chmod -R g=rwX dir1	Adds group read/write permission to dir1 and everything within it, and group execute permission on files or directories where someone has execute permission
chgrp testgrp file1	Sets file1's group to testgrp, if the user is a member of that group
chmod u+s file1	Sets the setuid bit on file1. (Doesn't change execute bit.)

Octal Notation

- Previous slide's syntax is nice for simple cases, but bad for complex changes
  - Alternative is octal notation, i.e., three or four digits from 0 to 7
- Digits from left (most significant) to right(least significant):
  [special bits][user bits][group bits][other bits]
- Special bit digit =
  (4 if setuid) + (2 if setgid) + (1 if sticky)
- All other digits =
  (4 if readable) + (2 if writable) + (1 if executable)

Octal Notation Examples

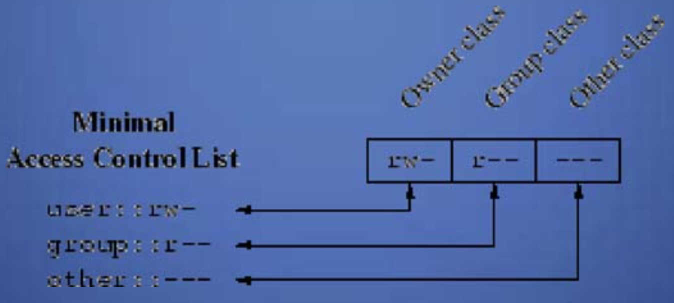
644 or 0644	read/write for owner, read-only for everyone else
775 or 0775	read/write/execute for owner and group, read/execute for others
640 or 0640	read/write for owner, read-only for group, forbidden to others
2775	same as 775, plus setgid (useful for directories)
777 or 0777	read/write/execute to everyone *(dangerous!)*
1777	same as 777, plus sticky bit

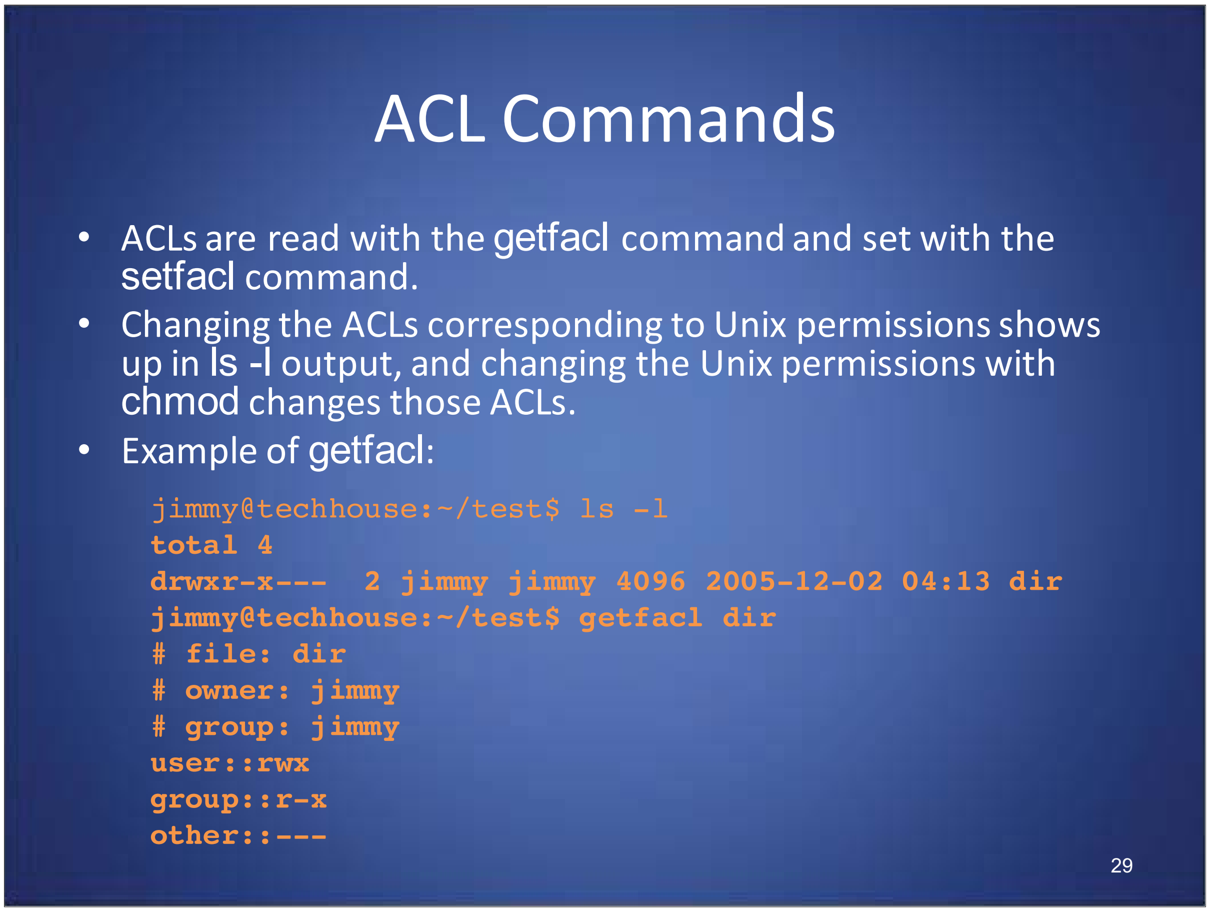
Limitations of Unix Permissions

- Unix permissions are not perfect
  - Groups are restrictive
  - Limitations on file creation
- Linux optionally uses POSIX ACLs
  - Builds on top of traditional Unix permissions
  - Several users and groups can be named in ACLs, each with different permissions
  - Allows for finer-grained access control
- Each ACL is of the form *type*:[*name*]:*rwx*
  - Setuid, setgid, and sticky bits are outside the ACL system

Minimal ACLs

- In a file with minimal ACLs, *name* does not appear, and the ACLs with *type* “user” and “group” correspond to Unix user and group permissions, respectively.
  - When name is omitted from a “user” type ACL entry, it applies to the file owner.

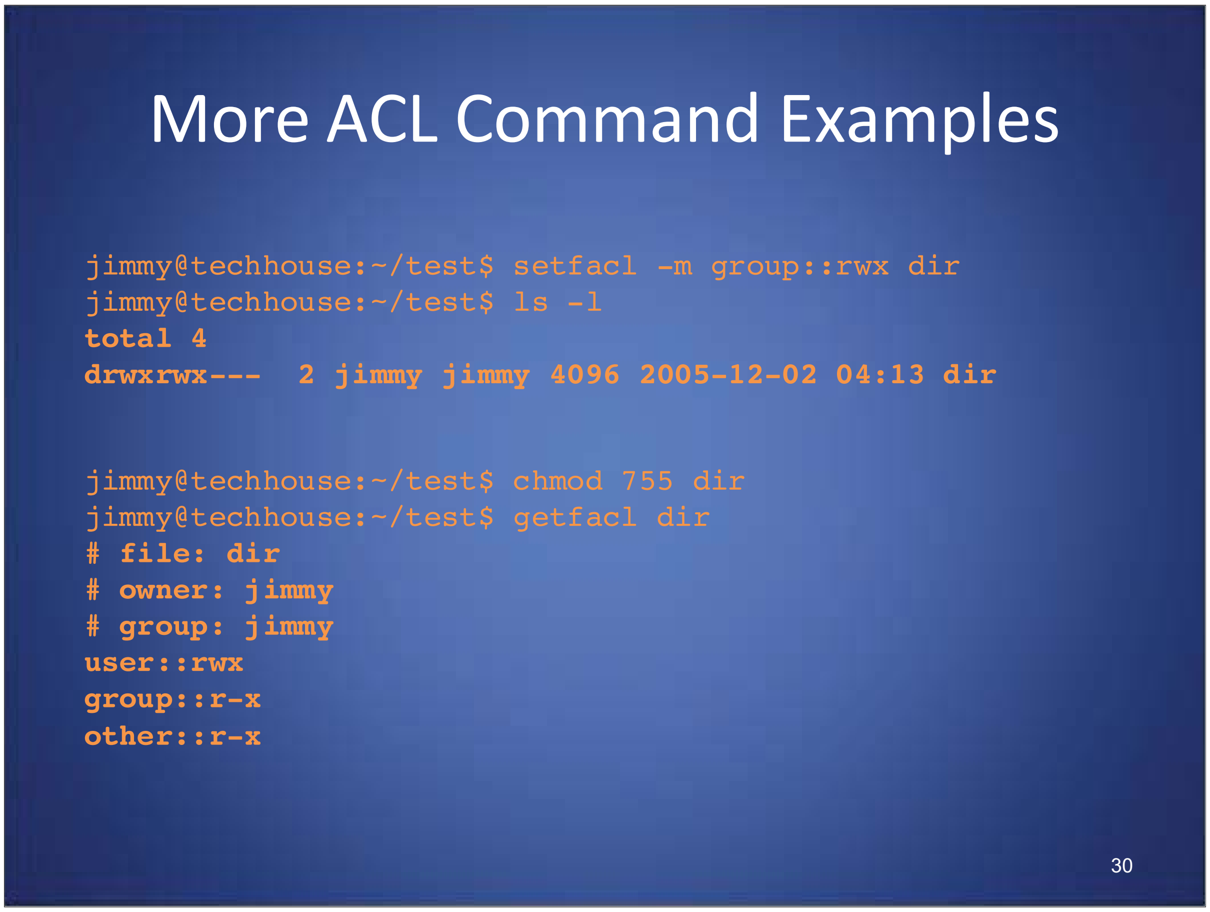




ACL Commands

- ACLs are read with the getfacl command and set with the setfacl command.
- Changing the ACLs corresponding to Unix permissions shows up in ls -l output, and changing the Unix permissions with chmod changes those ACLs.
- Example of getfacl:

```
jimmy@techhouse:~/test$ ls -l
total 4
drwxr-x---  2 jimmy jimmy 4096 2005-12-02 04:13 dir
jimmy@techhouse:~/test$ getfacl dir
# file: dir
# owner: jimmy
# group: jimmy
user::rwx
group::r-x
other::---
```



More ACL Command Examples

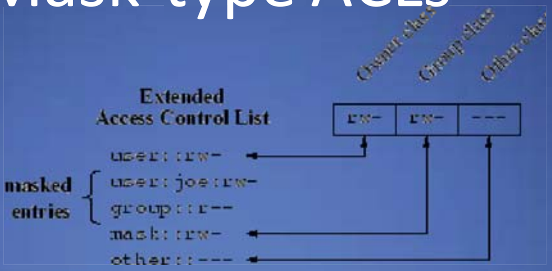
```
jimmy@techhouse:~/test$ setfacl -m group::rwx dir
jimmy@techhouse:~/test$ ls -l
total 4
drwxrwx---  2 jimmy jimmy 4096 2005-12-02 04:13 dir


jimmy@techhouse:~/test$ chmod 755 dir
jimmy@techhouse:~/test$ getfacl dir
# file: dir
# owner: jimmy
# group: jimmy
user::rwx
group::r-x
other::r-x
```

Extended ACLs

- ACLs that say more than Unix permissions are extended ACLs
  - Specific users and groups can be named and given permissions via ACLs, which fall under the group class (even for for ACLs naming users and not groups)
- With extended ACLs, mapping to and from Unix permissions is a bit complicated.
- User and other classes map directly to the corresponding Unix permission bits
- Group class contains named users and groups as well as owning group permissions. How to map?

Mask-type ACLs



- Unix group permissions now map to an ACL of *type* “mask”, which is an upper bound on permissions for all group class ACLs.
- All group class ACLs are logically and-ed with the mask before taking effect
  - rw-—xrw- & r-x—x--- = r----x--
- The ACL of *type* “group” with no *name* still refers to the Unix owning group
- Mask ACLs are created automatically with the necessary bits such that they do not restrict the other ACLs at all, but this can be changed

Extended ACL Example

```
jimmy@techhouse:~/test$ ls -l
total 4
drwxr-xr-x  2 jimmy jimmy 4096 2005-12-02 04:13 dir
jimmy@techhouse:~/test$ setfacl -m user:joe:rwx dir
jimmy@techhouse:~/test$ getfacl dir
# file: dir
# owner: jimmy
# group: jimmy
user::rwx
user:joe:rwx
group::r-x
mask::rwx
other::r-x

jimmy@techhouse:~/test$ ls -l
total 8
drwxrwxr-x+ 2 jimmy jimmy 4096 2005-12-02 04:13 dir
```

Extended ACL Example Explained

- The preceding slide grants the named user `joe` read, write, and execute access to `dir`.
  - `dir` now has extended rather than minimal ACLs.
- The mask is set to rwx, the union of the two group class ACLs (named user `joe` and the owning group).
- In `ls -l` output, the group permission bits show the mask, not the owning group ACL
  - Effective owning group permissions are the logical and of the owning group ACL and the mask, which still equals r-x.
  - This could reduce the effective owning group permissions if the mask is changed to be more restrictive.
- The + in the `ls -l` output after the permission bits indicates that there are extended ACLs, which can be viewed with `getfacl`.

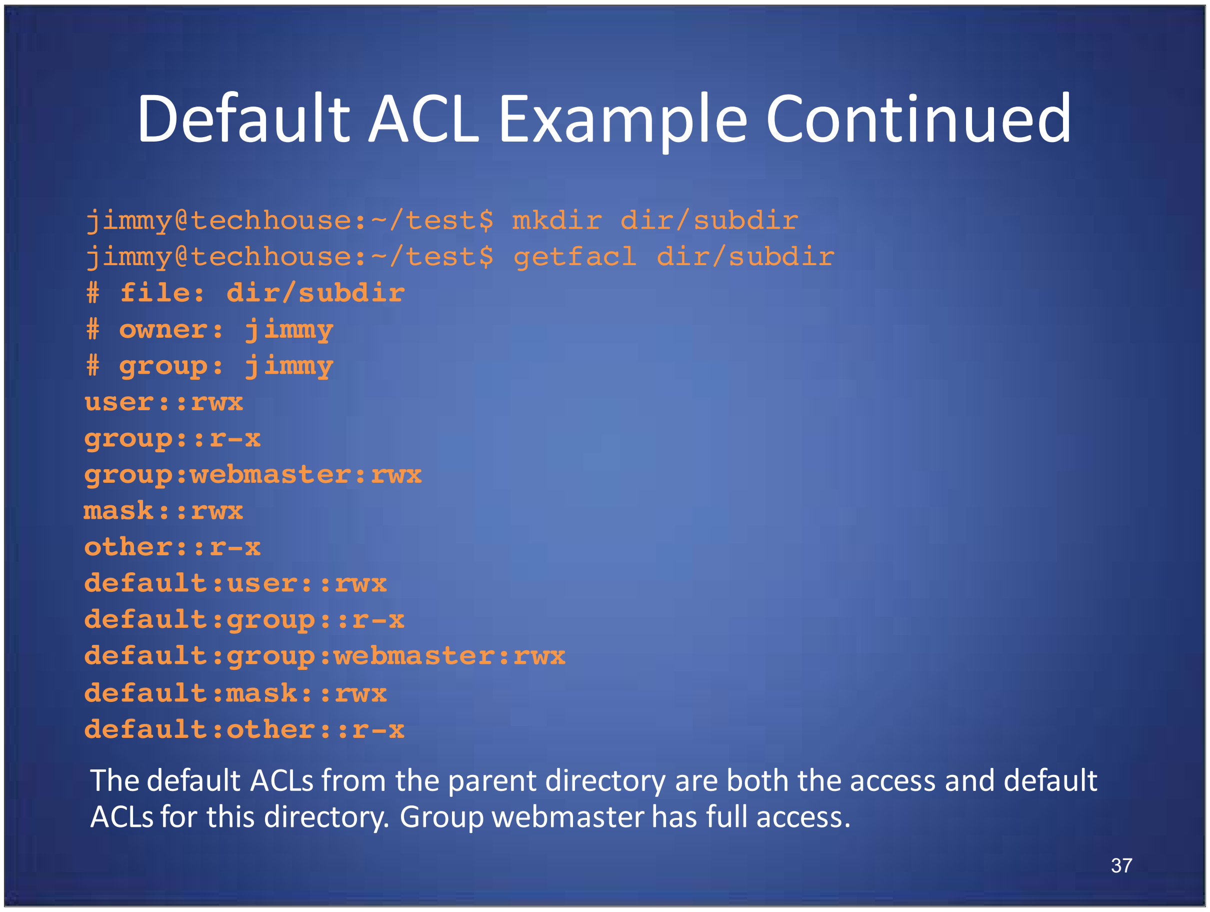
Default ACLs

- The kind of ACLs we've mentioned so far are access ACLs.
- A directory can have an additional set of ACLs, called default ACLs, which are inherited by files and subdirectories created within that directory.
  - Subdirectories inherit the parent directory's default ACLs as both their default and their access ACLs.
  - Files inherit the parent directory's default ACLs only as their access ACLs, since they have no default ACLs.
- The inherited permissions for the user, group, and other classes are logically and-ed with the traditional Unix permissions specified to the file creation procedure.

Default ACL Example

```
jimmy@techhouse:~/test$ setfacl -d -m group:webmaster:rwx
dir
jimmy@techhouse:~/test$ getfacl dir
# file: dir
# owner: jimmy
# group: jimmy
user::rwx
user:joe:rwx
group::r-x
mask::rwx
other::r-x
default:user::rwx
default:group::r-x
default:group:webmaster:rwx
default:mask::rwx
default:other::r-x
```

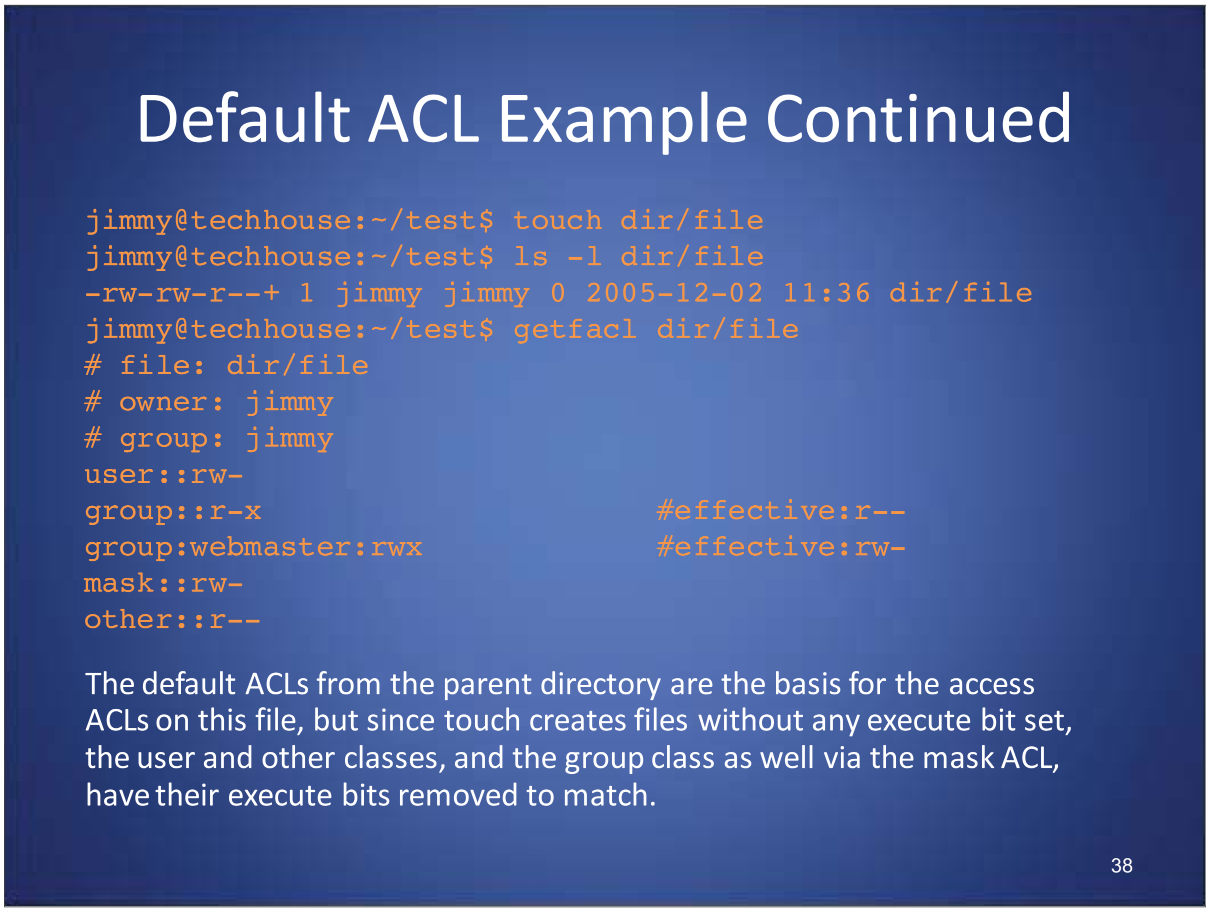
Note how this starts the default ACLs out as equal to the existing access ACLs plus the specified changes.



Default ACL Example Continued

```
jimmy@techhouse:~/test$ mkdir dir/subdir
jimmy@techhouse:~/test$ getfacl dir/subdir
# file: dir/subdir
# owner: jimmy
# group: jimmy
user::rwx
group::r-x
group:webmaster:rwx
mask::rwx
other::r-x
default:user::rwx
default:group::r-x
default:group:webmaster:rwx
default:mask::rwx
default:other::r-x
```

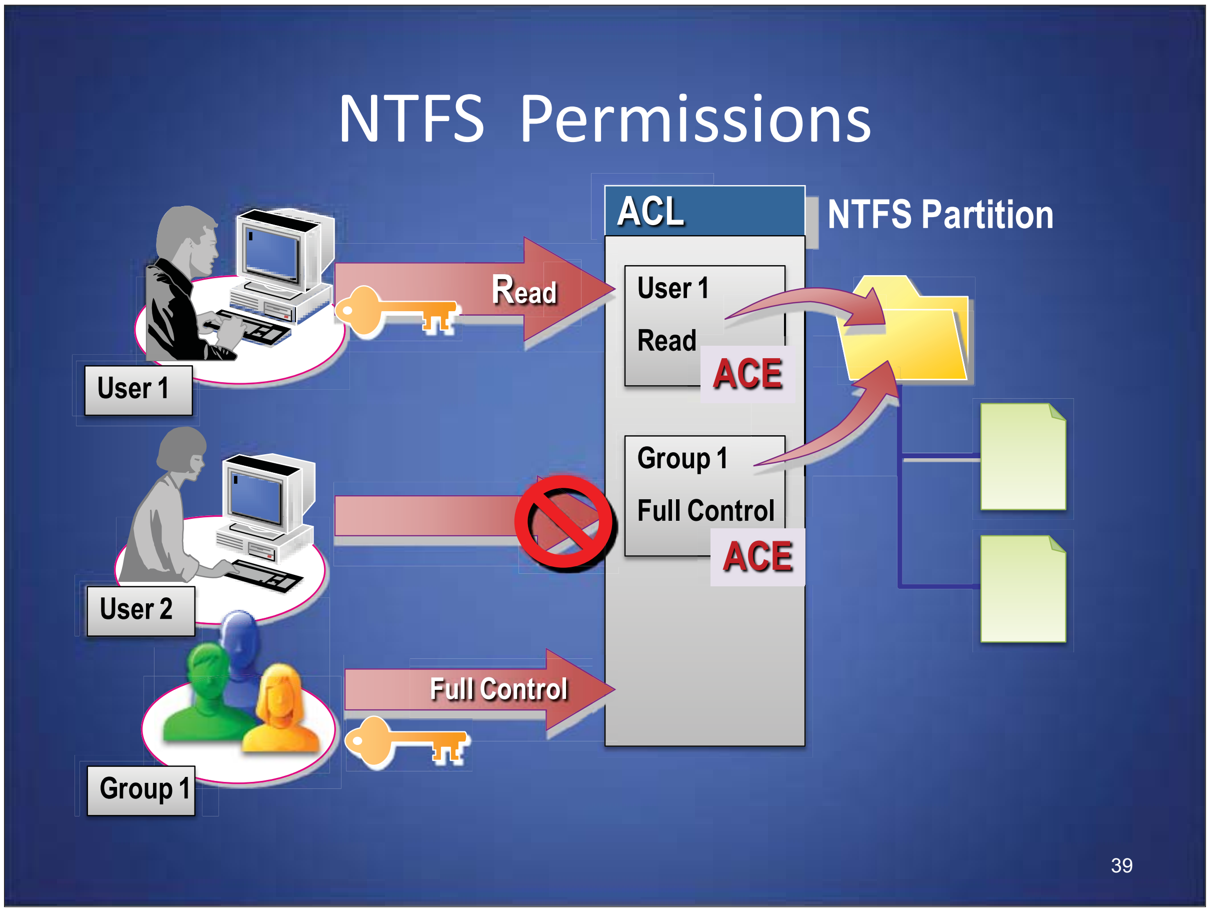
The default ACLs from the parent directory are both the access and default ACLs for this directory. Group webmaster has full access.



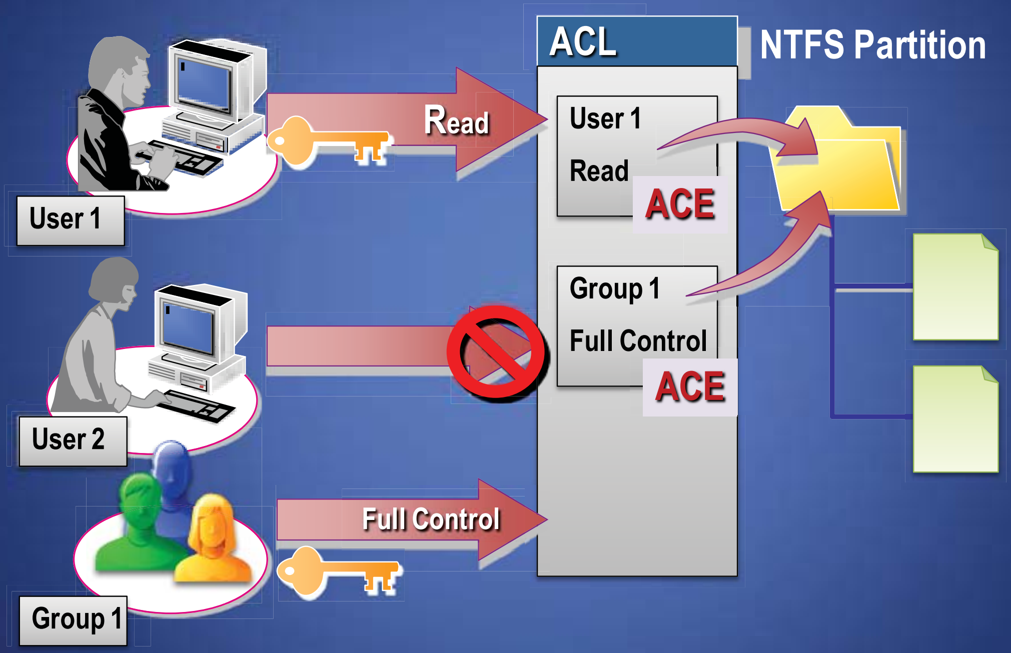
Default ACL Example Continued

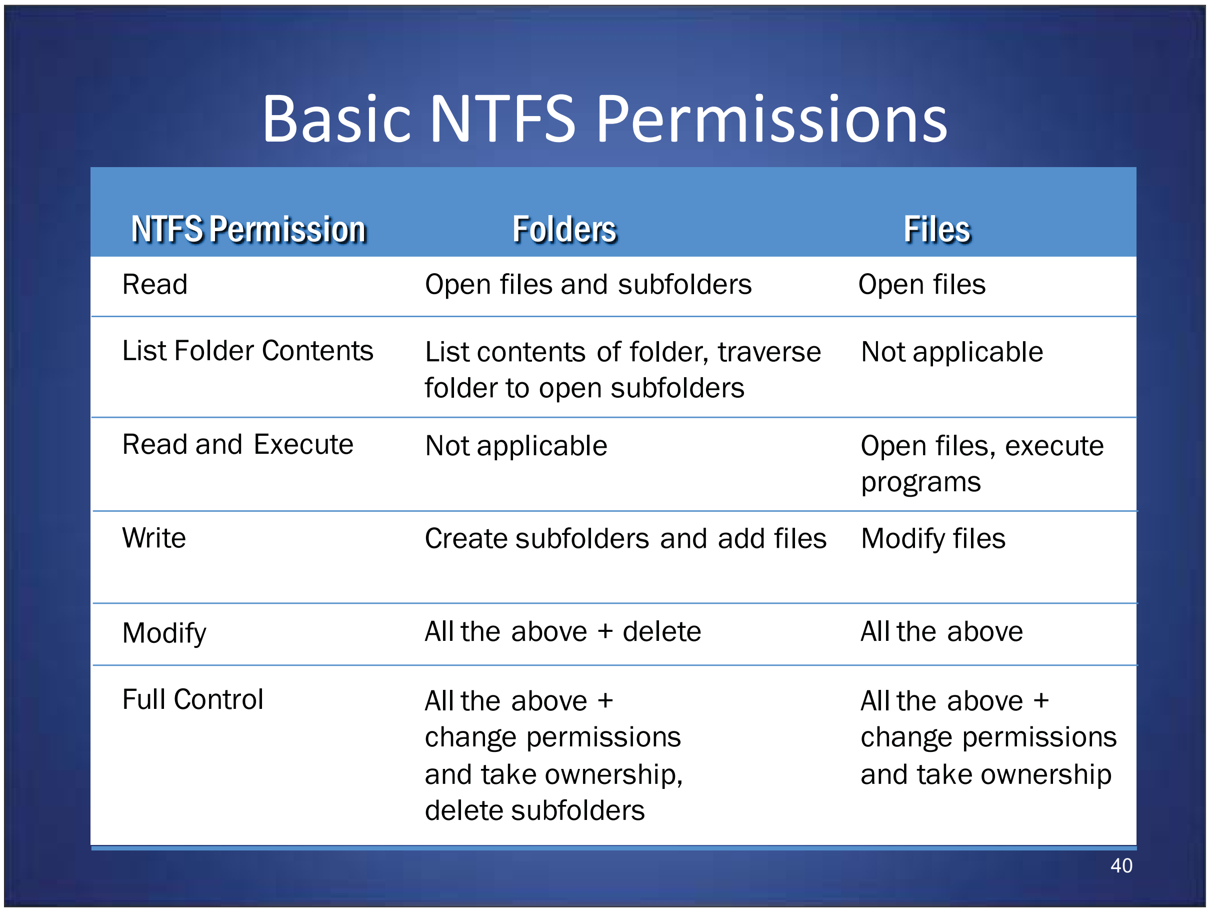
```
jimmy@techhouse:~/test$ touch dir/file
jimmy@techhouse:~/test$ ls -l dir/file
-rw-rw-r--+ 1 jimmy jimmy 0 2005-12-02 11:36 dir/file
jimmy@techhouse:~/test$ getfacl dir/file
# file: dir/file
# owner: jimmy
# group: jimmy
user::rw-
group::r-x                      #effective:r--
group:webmaster:rwx             #effective:rw-
mask::rw-
other::r--
```

The default ACLs from the parent directory are the basis for the access ACLs on this file, but since touch creates files without any execute bit set, the user and other classes, and the group class as well via the mask ACL, have their execute bits removed to match.



NTFS Permissions



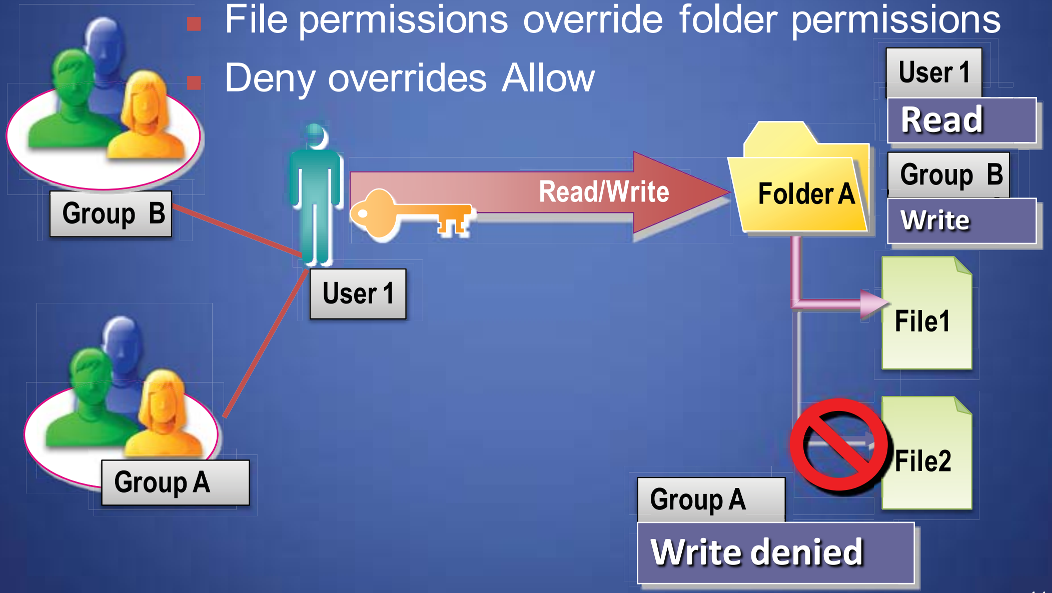


Basic NTFS Permissions

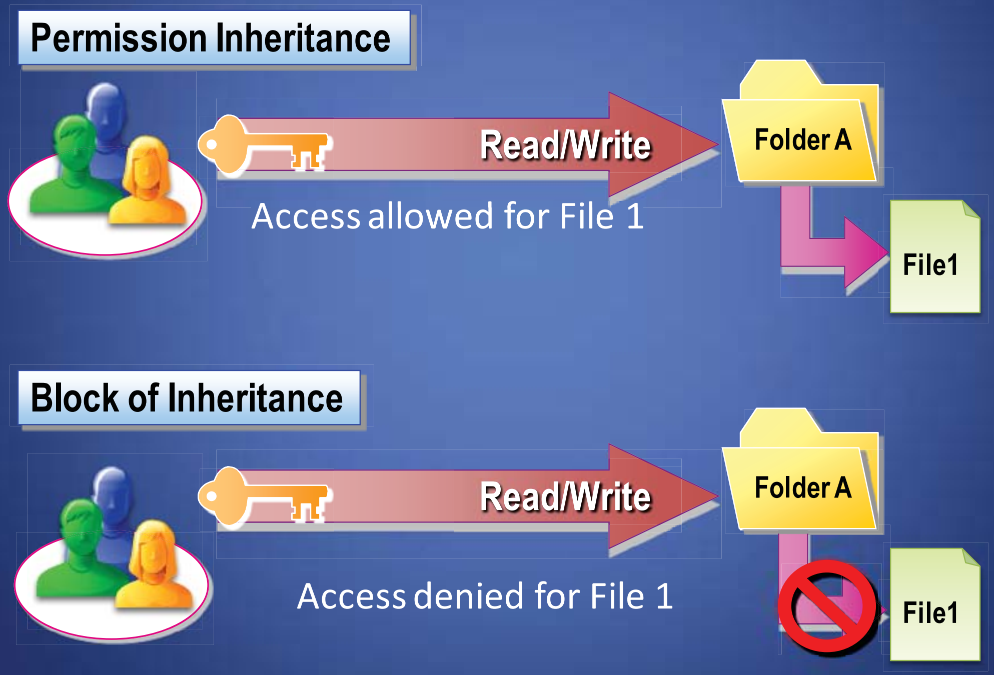
NTFS Permission	Folders	Files
Read	Open files and subfolders	Open files
List Folder Contents	List contents of folder, traverse folder to open subfolders	Not applicable
Read and Execute	Not applicable	Open files, execute programs
Write	Create subfolders and add files	Modify files
Modify	All the above + delete	All the above
Full Control	All the above + change permissions and take ownership, delete subfolders	All the above + change permissions and take ownership

Multiple NTFS permissions

- NTFS permissions are cumulative
- File permissions override folder permissions
- Deny overrides Allow



NTFS: permission inheritance

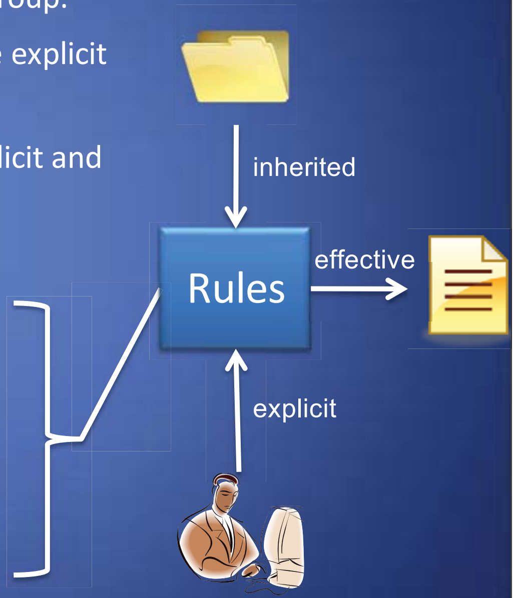


NTFS File Permissions

- **Explicit**: set by the *owner* for each user/group.
- **Inherited**: dynamically inherited from the explicit permissions of ancestor folders.
- **Effective**: obtained by combining the explicit and inherited permission.

Determining effective permissions:

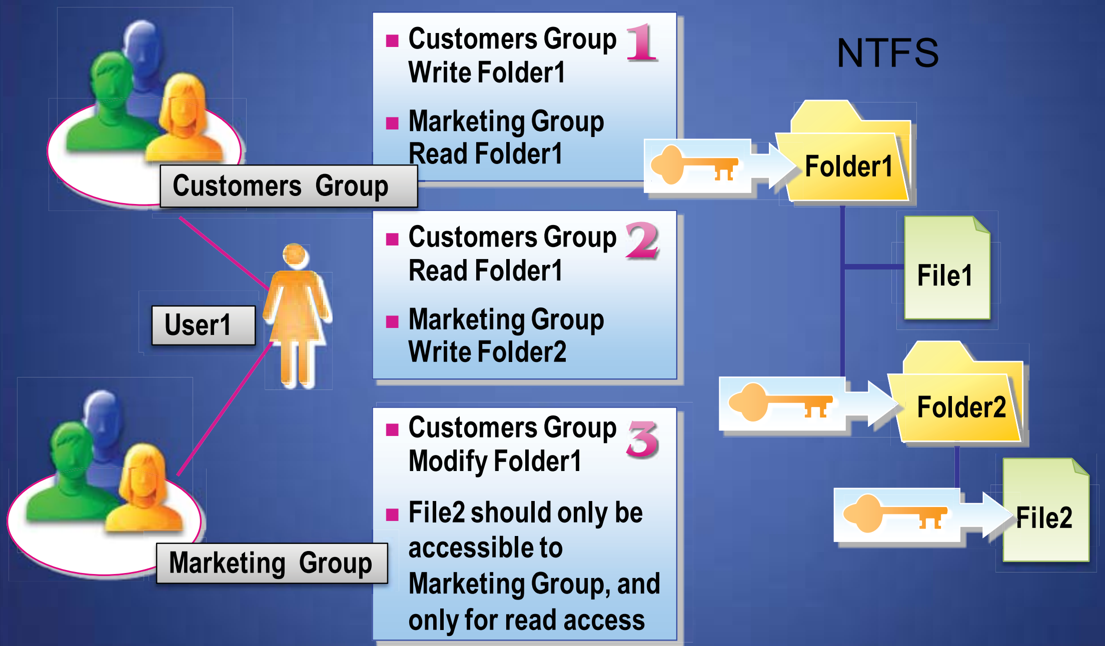
- By default, a user/group has no privileges.
- Explicit permissions override conflicting inherited permissions.
- Denied permissions override conflicting allowed permissions.



Access Control Algorithm

- The DACL of a file or folder is a sorted list of ACEs
  - Local ACEs precede inherited ACEs
  - ACEs inherited from folder F precede those inherited from parent of F
  - Among those with same source, Deny ACEs precede Allow ACEs
- Algorithm for granting access request (e.g., read and execute):
  - ACEs in the DACL are examined in order
  - Does the ACE refer to the user or a group containing the user?
  - If so, do any of the accesses in the ACE match those of the request?
  - If so, what type of ACE is it?
    - **Deny**: return ACCESS_DENIED
    - **Allow**: grant the specified accesses and if there are no remaining accesses to grant, return ACCESS_ALLOWED
  - If we reach the end of the DACL and there are remaining requested accesses that have not been granted yet, return ACCESS_DENIED

Example

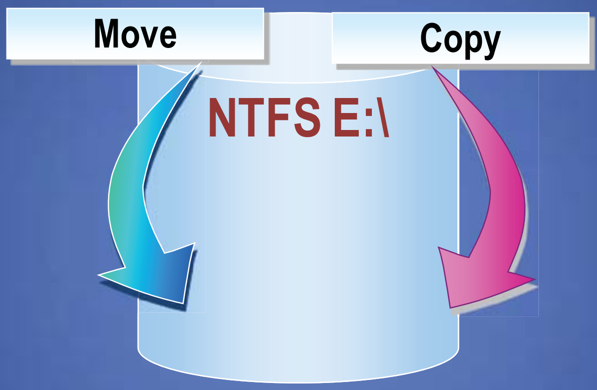


- Customers Group Write Folder1
- Marketing Group Read Folder1

- Customers Group Read Folder1
- Marketing Group Write Folder2

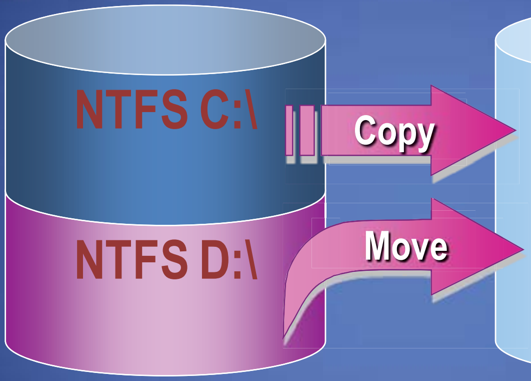
- Customers Group Modify Folder1
- File2 should only be accessible to Marketing Group, and only for read access

NTFS move vs. copy in same volume



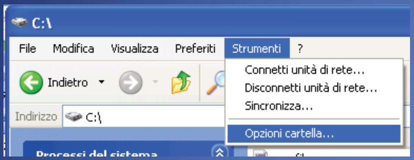
- If you **move** a file or a folder inside the same volume your permission will be the same of the **source** folder
- If you **copy** a file or a folder inside the same volume your permission will be the same of the **destination** folder

NTFS move vs. copy across volumes

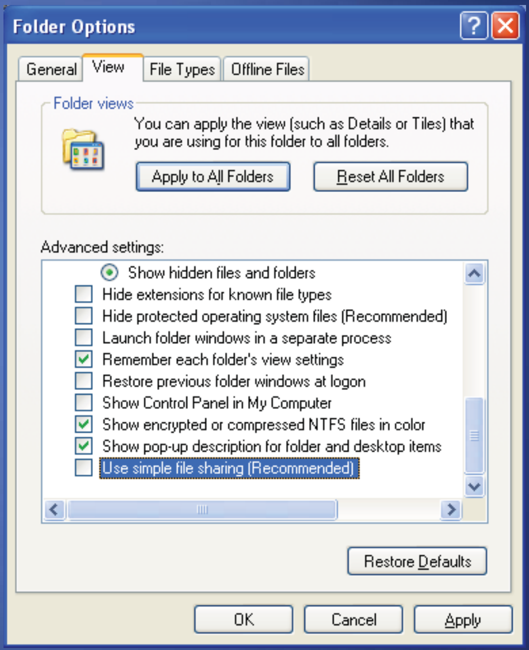


- If you **copy** or **move** a file or a folder on different volumes your permission will be the same of the **destination** folder

Setting File Permissions in Win XP



- NTFS permissions in Windows XP Pro are disabled by default.
- Using **Folder Options…** from **Tools** menu inside **Windows Explorer** is possible to activate NTFS permission in windows by unchecking **Use simple file sharing**



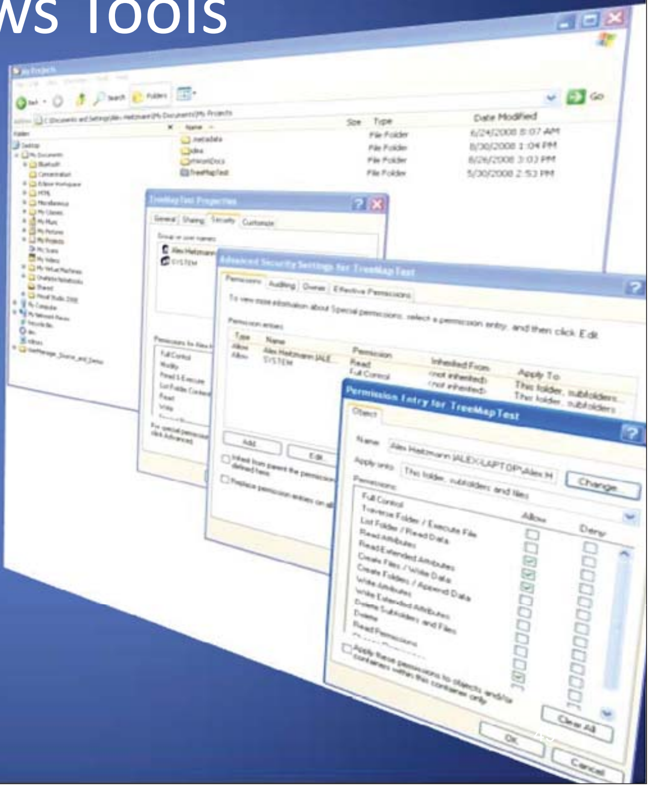
You can apply the view (such as Details or Tiles) that you are using for this folder to all folders.

Advanced settings:

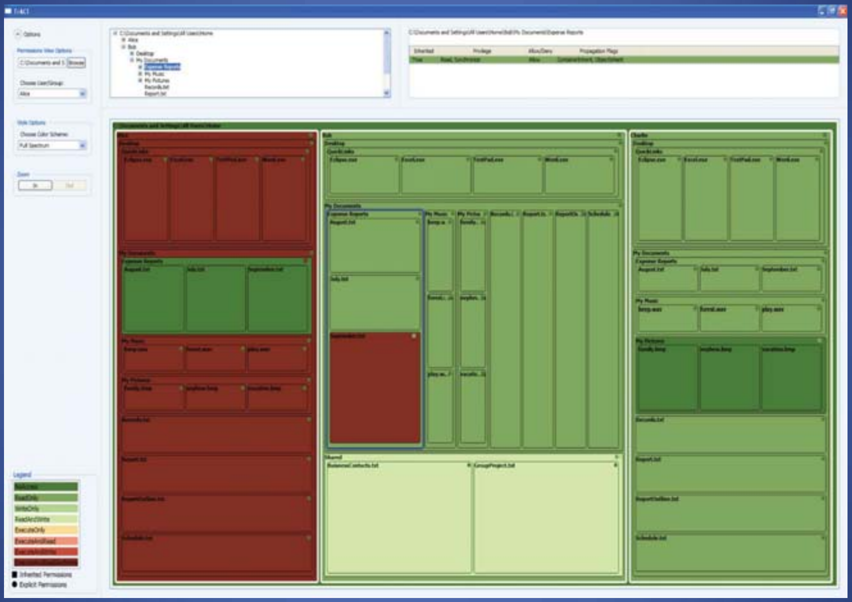
- ● Show hidden files and folders
- ☐ Hide extensions for known file types
- ☐ Hide protected operating system files (Recommended)
- ☐ Launch folder windows in a separate process
- ✓ Remember each folder's view settings
- ☐ Restore previous folder windows at logon
- ☐ Show Control Panel in My Computer
- ✓ Show encrypted or compressed NTFS files in color
- ✓ Show pop-up description for folder and desktop items
- ☐ Use simple file sharing (Recommended)

Windows Tools

- Access control management tools provide detailed information and controls, across multiple dialogs.
- Focus on single file/folders.
- It is challenging for an inexperienced user, or a system administrator dealing with very large file structures, to gain a global view of permissions within the file system



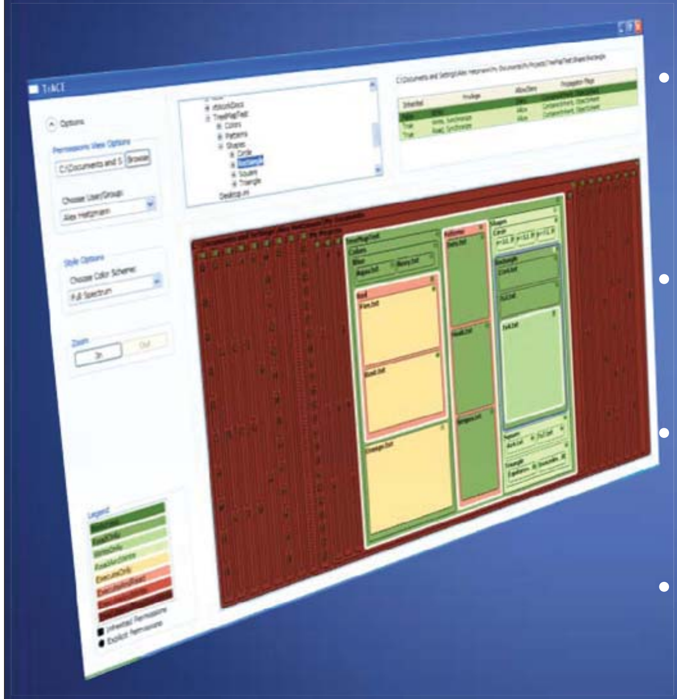
Treemap Access Control Evaluator (TrACE)



Sponsors:

Alexander Heitzmann, Bernardo Palazzi, Charalampos Papamanthou, Roberto Tamassia. *Effective Visualization of File System Access Control*, VizSEC 2008

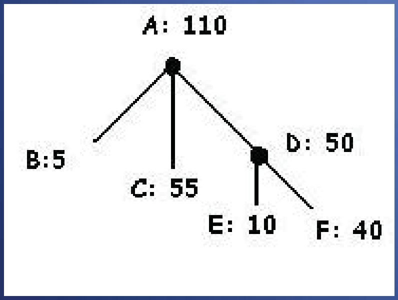
TrACE Highlights

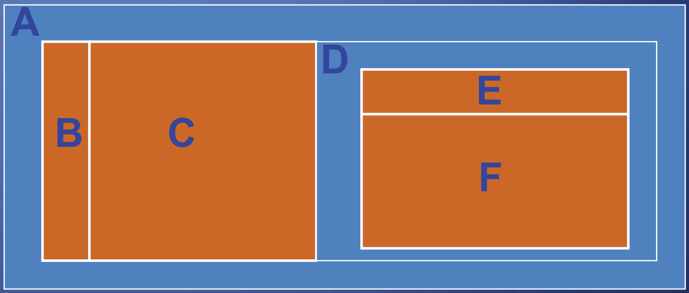


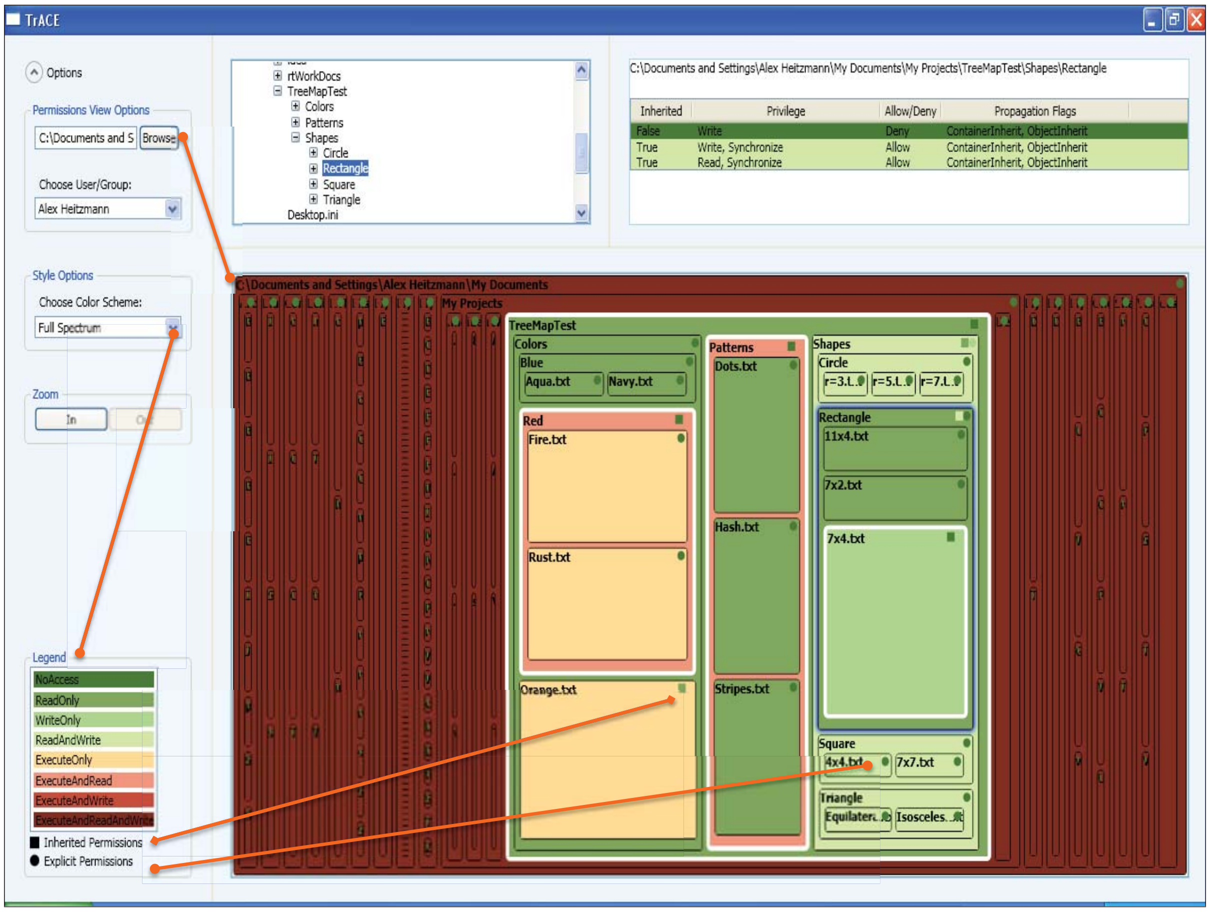
- At a glance, determine the explicit, inherited, and effective permissions of files and folders.
- Understand access control relationships between files and their ancestors
- Quickly evaluate large directory structures and find problem areas
- Layout based on treemaps

What is a Treemap?

- A visualization method to display large hierarchical data structures (trees)
- Layout based on nested rectangles.
- Treemaps were introduced by Ben Shneiderman in "Tree visualization with tree-maps: 2-d space-filling approach"; TOG 1991






TrACE

Options

Permissions View Options

C:\Documents and S Browse

Choose User/Group:

Alex Heitzmann

Style Options

Choose Color Scheme:

Full Spectrum

Zoom

In Out

C:\Documents and Settings\Alex Heitzmann\My Documents\My Projects\TreeMapTest\Shapes\Rectangle

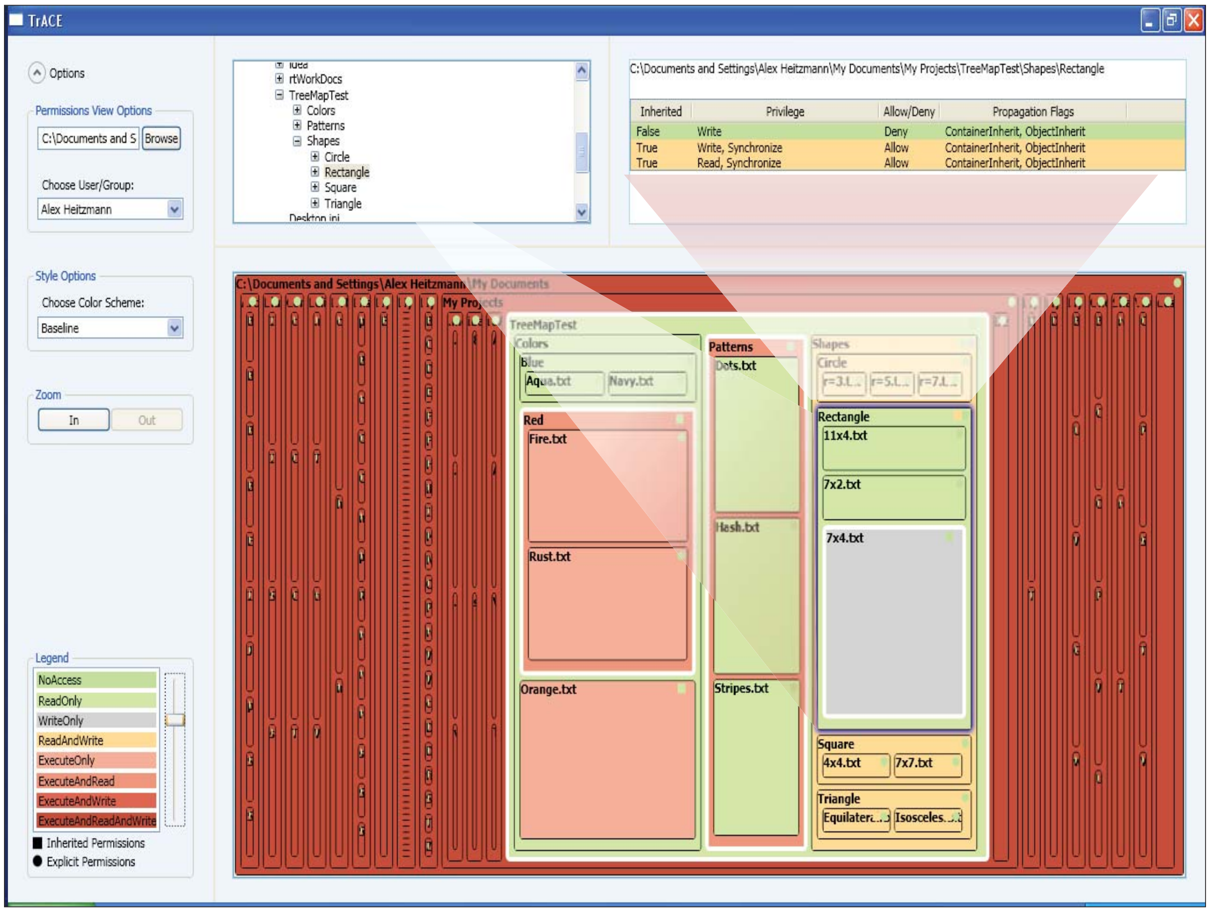
Inherited	Privilege	Allow/Deny	Propagation Flags
False	Write	Deny	ContainerInherit, ObjectInherit
True	Write, Synchronize	Allow	ContainerInherit, ObjectInherit
True	Read, Synchronize	Allow	ContainerInherit, ObjectInherit

Legend

NoAccess
ReadOnly
WriteOnly
ReadAndWrite
ExecuteOnly
ExecuteAndRead
ExecuteAndWrite
ExecuteAndReadAndWrite

■ Inherited Permissions

● Explicit Permissions


TrACE

Options

Permissions View Options

C:\Documents and S Browse

Choose User/Group:

Alex Heitzmann

Style Options

Choose Color Scheme:

Baseline

Zoom

In Out

C:\Documents and Settings\Alex Heitzmann\My Documents\My Projects\TreeMapTest\Shapes\Rectangle

Inherited	Privilege	Allow/Deny	Propagation Flags
False	Write	Deny	ContainerInherit, ObjectInherit
True	Write, Synchronize	Allow	ContainerInherit, ObjectInherit
True	Read, Synchronize	Allow	ContainerInherit, ObjectInherit

Legend

NoAccess
ReadOnly
WriteOnly
ReadAndWrite
ExecuteOnly
ExecuteAndRead
ExecuteAndWrite
ExecuteAndReadAndWrite

■ Inherited Permissions

● Explicit Permissions

Acknowledgment

- Much of these POSIX ACL slides are adapted (and some pictures are taken) from Andreas Grünbacher's paper *POSIX Access Control Lists on Linux*, available online at:

  http://www.suse.de/~agruen/acl/linux-acls/