

Access Control Lists

Don Porter
CSE 506

Background (1)

- ✦ If everything in Unix is a file...
 - ✦ Everything in Windows is an object
- ✦ Why not files?
 - ✦ Not all OS abstractions make sense as a file
- ✦ Examples:
 - ✦ Eject button on an optical drive
 - ✦ Network card

Windows object model

- ✦ Everything, including files, is represented as a generic OS object
- ✦ New object types can be created/extended with arbitrary methods beyond just open/read/write/etc.
- ✦ Objects are organized into a tree-like hierarchy
- ✦ Try out Windows object explorer (winobj)
 - ✦ Sysinternals.net

Background (2)

- ✦ A big goal for Windows NT and 2000 was centralizing workstation administration at companies/etc.
 - ✦ Create a user account once, can log onto all systems
 - ✦ Vs. creating different accounts on 100s of systems
- ✦ Active Directory: a Domain server that stores user accounts for the domain
 - ✦ Log on to a workstation using an AD account
 - ✦ Ex: CS\porter – Domain CS, user id porter
 - ✦ Used by CS department today, centralizes user management

Active Directory

- ✦ Centralized store of users, printers, workstations, etc.
- ✦ Each machine caches this info as needed
 - ✦ Ex., once you log in, the machine caches your credentials

Big Picture

- ✦ OSes need a “language” to express what is allowed and what isn't
- ✦ Access Control Lists are a common way to do this
- ✦ Structure: “Allowed | Denied: Subject Verb Object”

Unix permissions as ACLs

-rw-----@ 1 porter staff 151841 Nov 10 08:45 win2kacl.pdf

- ✦ Allowed | Denied: Subject Verb Object
- ✦ Allowed: porter read win2kacl.pdf
- ✦ Allowed: porter write win2kacl.pdf
- ✦ Denied: staff read win2kacl.pdf
- ✦ Denied: other * win2kacl.pdf

Fine-grained ACLs

- ✦ Why have subjects other than users/groups?
 - ✦ Not all of my programs are equally trusted
 - ✦ Web browser vs. tax returns
 - ✦ Want to run some applications in a restricted context
- ✦ Still want a unified desktop and file system
 - ✦ Don't want to log out and log in for different applications
- ✦ Real goal: Associate a restricted context with a program

Why different verbs/ objects

- ✦ Aren't read, write, and execute good enough?
- ✦ Example: Changing passwords
 - ✦ Yes, you read and write the password file
 - ✦ But not directly (since I shouldn't be able to change other passwords)
 - ✦ Really, the administrator gives a trusted utility/service permission to write entries
 - ✦ And gives you permission to call a specific service function (change password) with certain arguments (namely your own user id/pass)

Fine-grained access control lists

- ✦ Keep user accounts and associated permissions
 - ✦ But let users create restricted subsets of their permissions
- ✦ In addition to files, associate ACLs with any object
 - ✦ ACLs can be very long, with different rules for each user/context
- ✦ And not just RWX rules
 - ✦ But any object method can have different rules

Big picture

- ✦ ACLs are written in terms of enterprise-wide principals
 - ✦ Users in AD
 - ✦ Objects that may be system local or on a shared file system
 - ✦ Object types and verbs usually in AD as well
- ✦ ACLs are associated with a specific object, such as a file

Complete!

- ✦ Assertion: Any security policy you can imagine can be expressed using ACLs
 - ✦ Probably correct
- ✦ Challenges:
 - ✦ Correct enforcement of ACLs
 - ✦ Efficient enforcement of ACLs
 - ✦ Updating ACLs
 - ✦ Correctly writing the policies/ACLs in the first place

Correct enforcement

- ✦ Strategy: All policies are evaluated by a single function
- ✦ Implement the evaluation function once
 - ✦ Audit, test, audit, test until you are sure it looks ok
- ✦ Keep the job tractable by restricting the input types
- ✦ All policies, verbs, etc. have to be expressed in a way that a single function can understand
 - ✦ Shifts some work to application developer

Efficient enforcement

- ✦ Evaluating a single object's ACL is no big deal
- ✦ When context matters, the amount of work grows substantially
- ✦ Example: The Linux VFS checks permission starting at the current directory (or common parent), and traverses each file in the tree
 - ✦ Why?
 - ✦ To check the permissions that you should be allowed to find this file

Efficiency

- ✦ In addition to the file system, other container objects create a hierarchy in Windows
- ✦ Trade-off: Either check permissions from top-down on the entire hierarchy, or propagate updates
 - ✦ Linux: top-down traversal
 - ✦ Alternative: `chmod o-w /home/porter`
 - ✦ Walk each file under `/home/porter` and also drop other's write permission

Efficiency, cont

- ✦ AD decided the propagating updates was more efficient
- ✦ Intuition: Access checks are much more frequent than changes
 - ✦ Better to make the common case fast!

Harder than it looks

```
# ls /home/porter
drwxr-xr--x  porter porter 4096 porter
chmod o+r /home/porter/public
# chmod o-r porter
# ls /home/porter
drwxr-x---x  porter porter 4096 porter
```

Recursively change all children to o-r.
But do you change public?

Issues with propagating

- ✦ Need to distinguish between explicit and inherited changes to the child's permissions when propagating
 - ✦ Ex 1: If I take away read permission to my home directory, distinguish those files with an explicit read permission from those just inheriting from the parent
 - ✦ Ex 2: If I want to prevent the administrator from reading a file, make sure the administrator can't countermand this by changing the ACL on /home

AD's propagation solution

- ✦ When an ACL is explicitly changed, mark it as such
 - ✦ Vs. inherited permissions
- ✦ When propagating, delete and reapply inherited permissions
 - ✦ Leave explicit ACLs alone

Challenge: Policies to ACLs

- ✦ Assertion: Translating policies to ACLs is hard
- ✦ Hard to:
 - ✦ Express some policies as ACLs
 - ✦ Write the precise ACL you want
 - ✦ Identify all objects that you want to restrict
- ✦ Much research around developing policy languages that better balance: human usability and implementation correctness
 - ✦ This system strongly favors implementation correctness

Example Policy

- ✦ "Don't let this file leave the computer"
- ✦ Ideas?
 - ✦ Create a restricted process context that disables network access
 - ✦ Only give read permission to this context
- ✦ But, what if this process writes the contents to a new file? Or over IPC to an unrestricted process?
 - ✦ Does the ACL propagate with all output?
 - ✦ If so, what if the program has a legitimate need to access other data?

Summary

- ✦ Basic idea of ACL
- ✦ How it is used in Windows/AD
 - ✦ How extended for fine granularity
- ✦ Challenges with hierarchical enforcement, writing policies