

# L15: Control Access to Files



Hui Chen, Ph.D.  
Dept. of Engineering & Computer Science  
Virginia State University  
Petersburg, VA 23806

# Acknowledgement

---

- ❑ Many slides are from or are revised from the slides of the author of the textbook
  - Matt Bishop, Introduction to Computer Security, Addison-Wesley Professional, October, 2004, ISBN-13: 978-0-321-24774-5. [Introduction to Computer Security @ VSU's Safari Book Online subscription](#)
  - <http://nob.cs.ucdavis.edu/book/book-intro/slides/>

# Outline

---

- Access control lists
- Capability lists

# Access Control Lists

---

- ❑ Store **columns** of access control matrix with the object it represents to form a list of pairs, e.g.,

	File1	File2	File3
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

- File1: {(Andy, rx), (Betty, rwxo), (Charlie, rx)}
- File2: {(Andy, r), (Betty, r), (Charlie, rwo)}
- File3: {(Andy, rwo), (Charlie, w)}

# Definition

---

Let  $S$  be the set of subjects, and  $R$  the set of rights, of a system. An access control list (ACL)  $l$  is a set of pairs  $l = \{ (s, r) : s \in S, r \subseteq R \}$ . Let  $acl$  be a function that determines the access control list  $l$  associated with a particular object  $o$ . The interpretation of the access control list  $acl(o) = \{ (s_i, r_i) : 1 \leq i \leq n \}$  is that subject  $s_i$  may access  $o$  using any right in  $r_i$ .

# Default Permissions

---

- ❑ Normal: if not named, *no* rights over file
  - Principle of Fail-Safe Defaults
- ❑ If many subjects, may use groups or wildcards in ACL and given matched subjects default rights

# Default Permission: Example

---

## ❑ UNICOS 7.0

- ACL entries are (*user, group, rights*)
- If *user* is in *group*, has rights over file
- '\*' is wildcard for *user, group*
  - ❑ (holly, \*, r): holly can read file regardless of her group
  - ❑ (\*, gleep, w): anyone in group gleep can write file

# Abbreviations

---

- ❑ Combine subjects to make long access control lists short



# Abbreviations: Example

---

- ❑ Unix divides users into three classes
  - Owner of the file
  - Group owner of the file
  - All other users (the rest)
- ❑ Unix systems provides read (r), write (w), and execute (x) rights
- ❑ Unix then represents the permissions as three triplets
- ❑ Unix assigns ownership based on creating process
  - Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)

# Abbreviations: Discussion

---

- ❑ Suffer from a loss of granularity
- ❑ e.g., Unix system with 5 users
  - Anne wants to allow Beth to read her file, Caroline to write to it, Della to read and write to it, and Elizabeth to execute it.
  - Three triplets are insufficient to allow all desired modes of access
  - Cumbersome to express “everybody but user Fran”

# ACLs + Abbreviations

---

- ❑ Augment abbreviated lists with full-blown ACLs
  - Intent is to shorten ACL
- ❑ Use abbreviations as the default permission controls
- ❑ Explicit ACLs override abbreviations
- ❑ Exact method varies

# Example: IBM AIX

---

- ❑ Base permissions are abbreviations
- ❑ Extended permissions are ACLs with user, group
- ❑ ACL entries specify permissions to be added or deleted from the base permissions

# Permissions in IBM AIX

---

attributes:

base permissions

owner(bishop) : rw-

group(sys) : r--

others: ---

extended permissions enabled

specify rw- u:holly

permit -w- u:heidi, g=sys

permit rw- u:matt

deny -w- u:holly, g=faculty

# Creation and Maintenance of ACLs

---

## ❑ Some issues ...

- Which subjects can modify an object's ACL?
- If there is a privileged user (such as root in the UNIX system or administrator in Windows NT), do the ACLs apply to that user?
- Does the ACL support groups or wildcards (that is, can users be grouped into sets based on a system notion of “group” or on pattern matching)?
- How are contradictory access control permissions handled? If one entry grants read privileges only and another grants write privileges only, which right does the subject have over the object?
- If a default setting is allowed, do the ACL permissions modify it, or is the default used only when the subject is not explicitly mentioned in the ACL?

# ACL Modification

---

- ❑ Which subjects can modify an object's ACL?
  - Creator is given *own* right that allows this
  - System R provides a *grant* modifier (like a copy flag) allowing a right to be transferred, so ownership not needed
    - ❑ Transferring right to another modifies ACL

# Privileged Users

---

- ❑ Do ACLs apply to privileged users (*root*)?
  - Solaris: abbreviated lists do not, but full-blown ACL entries do
  - Other vendors: varies



# Groups and Wildcards

---

## ❑ Does the ACL support groups or wildcards?

- Classic form: no; in practice, usually
- e.g., AIX: base perms gave group sys read only

```
permit      -w-      u:heidi, g=sys
```

line adds write permission for heidi when in that group

## ■ e.g., UNICOS:

- ❑ holly : gleep : r
  - user holly in group gleep can read file
- ❑ holly : \* : r
  - user holly in any group can read file
- ❑ \* : gleep : r
  - any user in group gleep can read file

# Conflicts

---

- ❑ How are contradictory access control permissions handled?
  - Deny access if any entry would deny access
    - ❑ AIX: if any entry denies access, *regardless of rights given so far*, access is denied
  - Apply first entry matching subject
    - ❑ Cisco routers: run packet through access control rules (ACL entries) in order; on a match, stop, and forward the packet; if no matches, deny
      - Note default is deny so honors principle of fail-safe defaults

# Handling Default Permissions

---

- ❑ How are default permissions handled?
  - Apply ACL entry, and if none use defaults
    - ❑ Cisco router: apply matching access control rule, if any; otherwise, use default rule (deny)
  - Augment defaults with those in the appropriate ACL entry
    - ❑ AIX: extended permissions augment base permissions

# Revocation Question

---

- ❑ How do you remove subject's rights to a file?
  - Owner deletes subject's entries from ACL, or rights from subject's entry in ACL
- ❑ What if ownership not involved?
  - Depends on system
  - System R: restore protection state to what it was before right was given
    - ❑ May mean deleting descendent rights too ...

# Windows NT ACLs

---

## ❑ Different sets of rights

- Basic: read, write, execute, delete, change permission, take ownership
- Generic: no access, read (read/execute), change (read/write/execute/delete), full control (all), special access (assign any of the basics)
- Directory: no access, read (read/execute files in directory), list, add, add and read, change (create, add, read, execute, write files; delete subdirectories), full control, special access

# Icacs

---

```
C:\teaching\451>icaccls vigenere.m
vigenere.m NT AUTHORITY\SYSTEM:(I) (F)
          BUILTIN\Administrators:(I) (F)
          TL_HM302SC\hui:(I) (F)
          TL_HM302SC\demousr:(I) (F)
```

```
Successfully processed 1 files; Failed processing 0
files
```

```
C:\teaching\451>
```

# cacls

---

- ❑ Windows 8 and 10 reports, “NOTE: Cacs is now deprecated, please use Icacs.”

```
C:\teaching\451>cacls vigenere.m
```

```
C:\teaching\451\vigenere.m NT AUTHORITY\SYSTEM:(ID)F
                                BUILTIN\Administrators:(ID)F
                                TL_HM302SC\hui:(ID)F
                                TL_HM302SC\demousr:(ID)F
```

```
C:\teaching\451>
```

# icaccls /?

---

ICACLS preserves the canonical ordering of ACE entries:

- Explicit denials
- Explicit grants
- Inherited denials
- Inherited grants

perm is a permission mask and can be specified in one of two forms:

a sequence of simple rights:

- N - no access
- F - full access
- M - modify access
- RX - read and execute access
- R - read-only access
- W - write-only access
- D - delete access

a comma-separated list in parentheses of specific rights:

- DE - delete
- RC - read control
- WDAC - write DAC
- WO - write owner
- S - synchronize

AS - access system security

MA - maximum allowed

GR - generic read

GW - generic write

GE - generic execute

GA - generic all

RD - read data/list directory

WD - write data/add file

AD - append data/add subdirectory

REA - read extended attributes

WEA - write extended attributes

X - execute/traverse

DC - delete child

RA - read attributes

WA - write attributes

inheritance rights may precede either form and are applied

only to directories:

(OI) - object inherit

(CI) - container inherit

(IO) - inherit only

(NP) - don't propagate inherit

(I) - permission inherited from parent container



# Accessing Files

---

- ❑ User not in file's ACL nor in any group named in file's ACL: deny access
- ❑ ACL entry denies user access: deny access
- ❑ Take union of rights of all ACL entries giving user access: user has this set of rights over file

# Capability Lists

---

- ❑ Store **rows** of access control matrix with the object it represents to form a list of pairs, e.g.,

	File1	File2	File3
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }

# Semantics

---

## ❑ Like a bus ticket

- Mere possession indicates rights that subject has over object
- Object identified by capability (as part of the token)
  - ❑ Name may be a reference, location, or something else
- Architectural construct in capability-based addressing; this just focuses on protection aspects

## ❑ Must prevent process from altering capabilities

- Otherwise subject could change rights encoded in capability or object to which they refer

# Definition

---

Let  $O$  be the set of objects, and  $R$  the set of rights, of a system. A capability list  $c$  is a set of pairs  $c = \{ (o, r) : o \in O, r \subseteq R \}$ . Let  $\text{cap}$  be a function that determines the capability list  $c$  associated with a particular subject  $s$ . The interpretation of the capability list  $\text{cap}(s) = \{ (o_i, r_i) : 1 \leq i \leq n \}$  is that subject  $s$  may access  $o_i$  using any right in  $r_i$ .

# Implementation

---

## ❑ Tagged architecture

- Bits protect individual words
  - ❑ B5700: tag was 3 bits and indicated how word was to be treated (pointer, type, descriptor, *etc.*)

## ❑ Paging/segmentation protections

- Like tags, but put capabilities in a read-only segment or page
  - ❑ CAP system did this
- Programs must refer to them by pointers
  - ❑ Otherwise, program could use a copy of the capability—which it could modify

# Implementation

---

## ❑ Cryptography

- Associate with each capability a cryptographic checksum enciphered using a key known to OS
- When process presents capability, OS validates checksum
- Example: Amoeba, a distributed capability-based system
  - ❑ Capability is (*name, creating\_server, rights, check\_field*) and is given to owner of object
  - ❑ *check\_field* is 48-bit random number; also stored in table corresponding to *creating\_server*
  - ❑ To validate, system compares *check\_field* of capability with that stored in *creating\_server* table
  - ❑ ***Vulnerable if capability disclosed to another process***

# Amplifying

---

- ❑ Allows *temporary* increase of privileges
- ❑ Needed for modular programming
  - Module pushes, pops data onto stack  
`module stack ... endmodule.`
  - Variable *x* declared of type stack  
`var x: module;`
  - *Only* stack module can alter, read *x*
    - ❑ So process doesn't get capability, but needs it when *x* is referenced—a problem!
  - Solution: give process the required capabilities while it is in module

# Examples

---

## ❑ HYDRA: templates

- Associated with each procedure, function in module
- Adds rights to process capability *while the procedure or function is being executed*
- Rights deleted on exit

## ❑ Intel iAPX 432: access descriptors for objects

- These are really capabilities
- 1 bit in this controls amplification
- When ADT constructed, permission bits of type control object set to what procedure needs
- On call, if amplification bit in this permission is set, the above bits or'ed with rights in access descriptor of object being passed



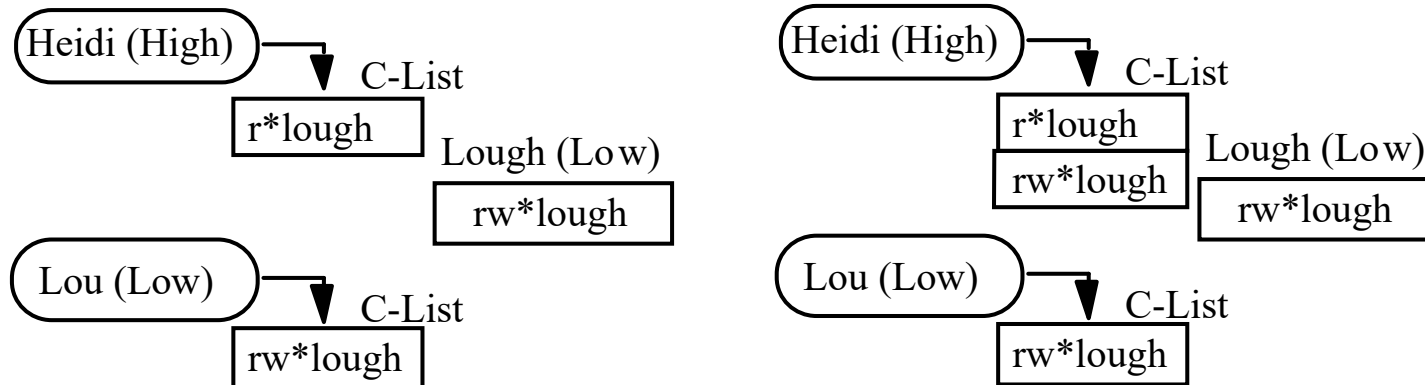
# Revocation

---

- ❑ Scan all capability-lists, remove relevant capabilities
  - Far too expensive!
- ❑ Use indirection
  - Each object has entry in a global object table
  - Names in capabilities name the entry, not the object
    - ❑ To revoke, zap the entry in the table
    - ❑ Can have multiple entries for a single object to allow control of different sets of rights and/or groups of users for each object
  - Example: Amoeba: owner requests server change random number in server table
    - ❑ All capabilities for that object now invalid

# Limits

## □ Problems if you do not control copying of capabilities



- The capability to write file *lough* is Low, and Heidi is High. So she reads (copies) the capability; now she can write to a Low file, violating the \*-property (of the Bell-Lapadula Model)!

# Remedies

---

- ❑ Label capability itself
  - Rights in capability depends on relation between its compartment and that of object to which it refers
    - ❑ In example, as as capability copied to High, and High dominates object compartment (Low), write right removed
- ❑ Check to see if passing capability violates security properties
  - In example, it does, so copying refused
- ❑ Distinguish between “read” and “copy capability”
  - Take-Grant Protection Model does this (“read”, “take”)

# ACLs vs. Capabilities

---

- ❑ Both theoretically equivalent; consider 2 questions
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
    - ACLs answer second easily; Capability-Lists, first
- ❑ Suggested that the second question, which in the past has been of most interest, is the reason ACL-based systems more common than capability-based systems
  - As first question becomes more important (in incident response, for example), this may change

# Exercise L15-1

---

- [Question 2 in page 259 of the textbook](#)

# Summary

---

- ❑ Access control mechanisms provide controls for users accessing files
- ❑ Many different forms
  - ACLs
  - Capabilities
  - ACLs vs. Capabilities