Access Control
Overview

• Access Control Matrix Model
• Protection State Transitions
  – Commands
  – Conditional Commands
• Mechanisms
  – Access control lists
  – Capability lists
  – Locks and keys
  – Rings-based access control
  – Propagated access control lists
Overview

• Protection state of system
  – Describes current settings, values of system relevant to protection

• Access control matrix
  – Describes protection state precisely
  – Matrix describing rights of subjects
  – State transitions change elements of matrix
AC Matrix Description

- Subjects $S = \{ s_1, \ldots, s_n \}$
- Objects $O = \{ o_1, \ldots, o_m \}$
- Rights $R = \{ r_1, \ldots, r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_{x_1}, \ldots, r_{y_1} \}$ means subject $s_i$ has rights $r_{x_1}, \ldots, r_{y_1}$ over object $o_j$
Example 1

- Processes $p, q$
- Files $f, g$
- Rights $r, w, x, a, o$

<table>
<thead>
<tr>
<th></th>
<th>$f$</th>
<th>$g$</th>
<th>$p$</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>rwo</td>
<td>$r$</td>
<td>rwxo</td>
<td>$w$</td>
</tr>
<tr>
<td>$q$</td>
<td>$a$</td>
<td>ro</td>
<td>$r$</td>
<td>rwxo</td>
</tr>
</tbody>
</table>
Example 2

- Procedures $inc\_ctr$, $dec\_ctr$, $manage$
- Variable $counter$
- Rights $+$, $-$, $call$

<table>
<thead>
<tr>
<th></th>
<th>counter</th>
<th>inc_ctr</th>
<th>dec_ctr</th>
<th>manage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$inc_ctr$</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$dec_ctr$</td>
<td>−</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$manage$</td>
<td></td>
<td>call</td>
<td>call</td>
<td>call</td>
</tr>
</tbody>
</table>
State Transitions

- Change the protection state of system
- $\mid \tau \mid$ represents transition
  - $X_i \mid \tau X_{i+1}$: command $\tau$ moves system from state $X_i$ to $X_{i+1}$
  - $X_i \mid \ast X_{i+1}$: a sequence of commands moves system from state $X_i$ to $X_{i+1}$
- Commands often called *transformation procedures*
Primitive Ops

• **create subject** \( s \); **create object** \( o \)
  – Creates new row, column in ACM; creates new column in ACM

• **destroy subject** \( s \); **destroy object** \( o \)
  – Deletes row, column from ACM; deletes column from ACM

• **enter** \( r \) **into** \( A[s, o] \)
  – Adds \( r \) rights for subject \( s \) over object \( o \)

• **delete** \( r \) **from** \( A[s, o] \)
  – Removes \( r \) rights from subject \( s \) over object \( o \)
Creating File

• Process $p$ creates file $f$ with $r$ and $w$ permission

```
  command create file(p, f)
  create object f;
  enter own into A[p, f];
  enter r into A[p, f];
  enter w into A[p, f];
  end
```
Mono-operational Commands

• Make process $p$ the owner of file $g$

  command  make·owner($p$, $g$)
  enter own into A[$p$, $g$];
  end

• Mono-operational command
  — Single primitive operation in this command
Conditional Commands

- Let \( p \) give \( q \) \( r \) rights over \( f \), if \( p \) owns \( f \)

\[
\text{command} \quad \text{grant} \cdot \text{read} \cdot \text{file} \cdot 1(p, f, q) \\
\text{if own in} \quad A[p, f] \\
\text{then} \\
\quad \text{enter} \quad r \quad \text{into} \quad A[q, f]; \\
\text{end}
\]

- Mono-conditional command
  - Single condition in this command
Multiple Conditions

• Let $p$ give $q$ $r$ and $w$ rights over $f$, if $p$ owns $f$ and $p$ has $c$ rights over $q$

```plaintext
command grant•read•file•2(p, f, q)
  if own in A[p, f] and c in A[p, q]
  then
    enter $r$ into A[q, f];
    enter $w$ into A[q, f];
  end
```
Copy Right

• Allows possessor to give rights to another
• Often attached to a right, so only applies to that right
  – $r$ is read right that cannot be copied
  – $rc$ is read right that can be copied
• Is copy flag copied when giving $r$ rights?
  – Depends on model, instantiation of model
Own Right

• Usually allows possessor to change entries in corresponding AC Matrix column
  – So owner of object can add, delete rights for others
  – May depend on what system allows
    • Can’t give rights to specific (set of) users
    • Can’t pass copy flag to specific (set of) users
Attenuation of Privilege

• Intuitive principle says *you can’t give rights you do not possess*
  – Restricts addition of rights within a system
  – Usually *ignored* for owner
    • Why? Mostly owner can grant herself any rights!
AC Safety

• System AC Safety
  – Start with access control matrix $A$
  – *Leak*: commands can add right $r$ to an element of $A$ not containing $r$
  – *Safe*: System is *safe with respect to $r$* if $r$ cannot be leaked

• Are algorithms *implemented correctly*?
Example: File System

• Superuser has access to all files
• Users have access to own files
• What is Safety here?
  – only user A can authenticate as user A
  – no “change mode”, “change owner” commands
  – only superuser can get superuser privileges
• Question: how useful is “safety”?  
  – doesn’t differentiate leaks vs. authorized transfers
  – solution: “trust” framework
(Un)decidability of Safety

- Given initial state $X_0 = (S_0, O_0, A_0)$, set of primitive commands $c$, can we reach a state $X_n$ where $\exists s, o$ such that $A_n[s, o]$ includes a right $r$ not in $A_0[s, o]$? (is a rights leak possible?)

- **Decidability:** Given a system where each command consists of a single primitive command (mono-operational), there exists an algorithm that will determine if a protection system with initial state $X_0$ is safe with respect to right $r$.

- **Undecidability:** For a given state of an arbitrary protection system the problem of determining if it is safe with respect to a given right is undecidable (proof: halting problem, “leak” = halting state).

AC Mechanisms

• Access control lists
• Capabilities
• Locks and keys
• Rings-based access control
• Propagated access control lists
Access Control Lists

- **Columns** of access control matrix

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
<th>file3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>rx</td>
<td>r</td>
<td>rwo</td>
</tr>
<tr>
<td>Betty</td>
<td>rwxo</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>rx</td>
<td>rwo</td>
<td>w</td>
</tr>
</tbody>
</table>

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

• ACLs can be very long!
• Idea: combine users
  – UNIX: 3 classes of users: owner, group, rest
  – rwx rwx rwx
    rest
    group
    owner
  – Ownership assigned based on creating process
    • Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)
ACLs + Abbreviations

• Augment abbreviated lists with ACLs
  – Intent is to shorten ACL

• ACLs override abbreviations
  – Exact method varies

• Example: IBM AIX
  – Base permissions are abbreviations, extended permissions are ACLs with user, group
  – ACL entries can add rights, but on deny, access is denied
Example: 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
ACL Modifications

• Who can do this?
  – 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

- Classic form: no; in practice, usually
  - AIX: base perms gave group sys read only
    \[
    \text{permit} \ -w- \ u:heidi, \ g=sys
    \]
    line adds write permission for heidi when in that group
  - 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

• Deny access if any entry would deny access
  – AIX: if any entry denies access, *regardless or 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 for *fail-safe defaults*
Default Permissions

• 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

• 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 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
Enforcement: 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

- **Rows** of access control matrix

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
<th>file3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>rx</td>
<td>r</td>
<td>rwo</td>
</tr>
<tr>
<td>Betty</td>
<td>rwxo</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>rx</td>
<td>rwo</td>
<td>w</td>
</tr>
</tbody>
</table>

C-Lists:
- Andy: { (file1, rx) (file2, r) (file3, rwo) }
- Betty: { (file1, rwxo) (file2, r) }
- Charlie: { (file1, rx) (file2, rwo) (file3, w) }
Meaning of Capabilities

• “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
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 (cont’d)

• Cryptography
  – Associate with each capability a cryptographic checksum encrypted 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**
Question

• Bad guy: why not simply copy capability?
  – *What can the OS do to prevent this?*
Amplification

• *temporary* elevation/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 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 (ADT = access descriptor)
  – 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 / Deletion of Rights

• Scan all C-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
• Problems if you don’t 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! (Bell-LaPadula)
Remedies

• Label capability itself
  – Rights in capability depends on relation between its compartment and that of object to which it refers
    • In example, 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; C-Lists, first

• second question has been of most interest in the past thus ACL-based systems more common than capability-based systems
  – As first question becomes more important (in incident response, for example), this may change
Locks and Keys

- Associate information (*lock*) with object, information (*key*) with subject
  - Latter controls what the subject can access and how
  - Subject presents key; if it corresponds to any of the locks on the object, access granted

- This can be dynamic
  - ACLs, C-Lists static and must be manually changed
  - Locks and keys can change based on system constraints, other factors (not necessarily manual)
Cryptographic Implementation

• Enciphering with lock; deciphering with key
  – Encipher object \( o \); store \( E_k(o) \)
  – Use subject’s key \( k' \) to compute \( D_k(E_k(o)) \)
  – Any of \( n \) can access \( o \): store
    \[ o' = (E_1(o), ..., E_n(o)) \]
  – Requires consent of all \( n \) to access \( o \): store
    \[ o' = (E_1(E_2(...(E_n(o))...)) \]
Example: IBM

• IBM 370: process gets access key; pages get storage key and fetch bit
  – Fetch bit clear: read access only
  – Fetch bit set, access key 0: process can write to (any) page
  – Fetch bit set, access key matches storage key: process can write to page
  – Fetch bit set, access key non-zero and does not match storage key: no access allowed
Example: Cisco Router

• **Dynamic access control lists**

```
access-list 100 permit tcp any host 10.1.1.1 eq telnet
access-list 100 dynamic test timeout 180 permit ip any host \
  10.1.2.3 time-range my-time
time-range my-time
  periodic weekdays 9:00 to 17:00
line vty 0 2
  login local
  autocommand access-enable host timeout 10
```

• **Limits external access to 10.1.2.3 to 9AM–5PM**
  – Adds temporary entry for connecting host once user supplies name, password to router
  – Connections good for 180 minutes
    • Drops access control entry after that
Type Checking

• Lock is type, key is operation
  – Example: UNIX system call write can’t work on directory object but does work on file
  – Example: split I&D space of PDP-11
  – Example: countering buffer overflow attacks on the stack by putting stack on non-executable pages/segments
    • Then code uploaded to buffer won’t execute
    • Does not stop other forms of this attack, though ...
Ring-based Access Control

- Process (segment) accesses another segment
  - Read
  - Execute
- Gate is an entry point for calling segment
- Rights:
  - r read
  - w write
  - a append
  - e execute
Reading/writing/appending

- Procedure executing in ring \( r \)
- Data segment with access bracket \( (a_1, a_2) \)
- Mandatory access rule
  - \(- r \leq a_1\) allow access
  - \(- a_1 < r \leq a_2\) allow \( r \) access; not \( w, a \) access
  - \(- a_2 < r\) deny all access
Executing

- Procedure executing in ring $r$
- Call procedure in segment with access bracket $(a_1, a_2)$ and call bracket $(a_2, a_3)$
  - Often written $(a_1, a_2, a_3)$
- Mandatory access rule
  - $r < a_1$ allow access; ring-crossing fault
  - $a_1 \leq r \leq a_2$ allow access; no ring-crossing fault
  - $a_2 < r \leq a_3$ allow access if through valid gate
  - $a_3 < r$ deny all access
Versions

• Multics
  – 8 rings (from 0 to 7)
• Digital Equipment’s VAX
  – 4 levels of privilege: user, monitor, executive, kernel
• Older systems
  – 2 levels of privilege: user, supervisor
• Today
  – Linux (2/3+ rings, depending on processor etc)
Propagated ACLs

• Propagated Access Control List
• Creator kept with PACL, copies
  – Only owner can change PACL
  – Subject reads object: object’s PACL associated with subject
  – Subject writes object: subject’s PACL associated with object
• Notation: PACLₙ means s created object; PACL(e) is PACL associated with entity e
Example with Multiple Creators

- Betty reads Ann’s file *dates*
  \[ \text{PACL}(\text{Betty}) = \text{PACL}_{\text{Betty}} \cap \text{PACL}(\text{dates}) = \text{PACL}_{\text{Betty}} \cap \text{PACL}_{\text{Ann}} \]

- Betty creates file *datescopy*
  \[ \text{PACL}(\text{datescopy}) = \text{PACL}_{\text{Betty}} \cap \text{PACL}_{\text{Ann}} \]

- PACL\(_{\text{Betty}}\) allows Cher to access objects, but PACL\(_{\text{Ann}}\) does not; both allow June to access objects
  - June can read *datescopy*
  - Cher cannot read *datescopy*

- Can be augmented by discretionary AC, e.g. ACLs
  - Betty decides Cher should not read *datescopy*
ACL vs. PACL

- **ACL**
  - associated with *object*
  - static, with object

- **PACL**
  - associated with *data*,
  - follows information flow
  - slower (implementation)
  - ORCON Policies
Key Points

- AC matrix - simple abstraction mechanism for representing protection state
  - 6 primitive operations alter matrix
  - transitions can be expressed as commands composed of these operations and, possibly, conditions
- AC mechanisms control users accessing resources
- Many different forms
  - ACLs, capabilities, locks and keys
    - Type checking too
  - Ring-based mechanisms
  - PACLs