Hardware Support

• Token-based
  – Used to compute response to challenge
    • May encipher or hash challenge
    • May require PIN from user

• Temporally-based
  – Every minute (or so) different number shown
    • Computer knows what number to expect when
  – User enters number and fixed password
C-R and Dictionary Attacks

• Same as for fixed passwords
  – Attacker knows challenge $r$ and response $f(r)$; if $f$ encryption function, can try different keys
  • May only need to know form of response; attacker can tell if guess correct by looking to see if deciphered object is of right form
  • Example: Kerberos Version 4 used DES, but keys had 20 bits of randomness; Purdue attackers guessed keys quickly because deciphered tickets had a fixed set of bits in some locations
Encrypted Key Exchange

- Defeats off-line dictionary attacks
- Idea: random challenges enciphered, so attacker cannot verify correct decipherment of challenge
- Assume Alice, Bob share secret password $s$
- In what follows, Alice needs to generate a random public key $p$ and a corresponding private key $q$
- Also, $k$ is a randomly generated session key, and $R_A$ and $R_B$ are random challenges
EKE Protocol

Alice $\| E_s(p)$ $\rightarrow$ Bob

Alice $\leftarrow E_s(E_p(k))$ $\rightarrow$ Bob

Now Alice, Bob share a randomly generated secret session key $k$

Alice $\rightarrow E_k(R_A)$ $\rightarrow$ Bob

Alice $\leftarrow E_k(R_A R_B)$ $\rightarrow$ Bob

Alice $\leftarrow E_k(R_B)$ $\rightarrow$ Bob
Biometrics

• Automated measurement of biological, behavioral features that identify a person
  – Fingerprints: optical or electrical techniques
    • Maps fingerprint into a graph, then compares with database
    • Measurements imprecise, so approximate matching algorithms used
  – Voices: speaker verification or recognition
    • Verification: uses statistical techniques to test hypothesis that speaker is who is claimed (speaker dependent)
    • Recognition: checks content of answers (speaker independent)
Other Characteristics

- Can use several other characteristics
  - Eyes: patterns in irises unique
    - Measure patterns, determine if differences are random; or correlate images using statistical tests
  - Faces: image, or specific characteristics like distance from nose to chin
    - Lighting, view of face, other noise can hinder this
  - Keystroke dynamics: believed to be unique
    - Keystroke intervals, pressure, duration of stroke, where key is struck
    - Statistical tests used
Cautions

• These can be fooled!
  – Assumes biometric device accurate in the environment it is being used in!
  – Transmission of data to validator is tamperproof, correct
Location

- If you know where user is, validate identity by seeing if person is where the user is
  - Requires special-purpose hardware to locate user
    - GPS (global positioning system) device gives location signature of entity
    - Host uses LSS (location signature sensor) to get signature for entity
Multiple Methods

• Example: “where you are” also requires entity to have LSS and GPS, so also “what you have”

• Can assign different methods to different tasks
  – As users perform more and more sensitive tasks, must authenticate in more and more ways (presumably, more stringently) File describes authentication required
    • Also includes controls on access (time of day, etc.), resources, and requests to change passwords

• Pluggable Authentication Modules
PAM

- Idea: when program needs to authenticate, it checks central repository for methods to use
- Library call: `pam_authenticate`
  - Accesses file with name of program in `/etc/pam_d`
- Modules do authentication checking
  - `sufficient`: succeed if module succeeds
  - `required`: fail if module fails, but all required modules executed before reporting failure
  - `requisite`: like `required`, but don’t check all modules
  - `optional`: invoke only if all previous modules fail
Example PAM File

```ini
auth sufficient  /usr/lib/pam_ftp.so
auth required    /usr/lib/pam_unix_auth.so use_first_pass
auth required    /usr/lib/pam_listfile.so onerr=succeed \
                   item=user sense=deny file=/etc/ftpusers
```

For ftp:

1. If user “anonymous”, return okay; if not, set PAM_AUTHTOK to password, PAM_RUSER to name, and fail

2. Now check that password in PAM_AUTHTOK belongs to that of user in PAM_RUSER; if not, fail

3. Now see if user in PAM_RUSER named in /etc/ftpusers; if so, fail; if error or not found, succeed
Description

<table>
<thead>
<tr>
<th>objects (entities)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$  $s_2$  $s_n$</td>
</tr>
<tr>
<td>$o_1$  $o_2$  $o_m$</td>
</tr>
</tbody>
</table>

- Subjects $S = \{ s_1, \ldots, s_n \}$
- Objects $O = \{ o_1, \ldots, o_m \}$
- Rights $R = \{ r_1, \ldots, r_k \}$
- Entries $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \ldots, r_y \}$ means subject $s_i$ has rights $r_x, \ldots, r_y$ 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><em>inc_ctr</em></td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>dec_ctr</em></td>
<td>−</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>manage</em></td>
<td></td>
<td><em>call</em></td>
<td><em>call</em></td>
<td><em>call</em></td>
</tr>
</tbody>
</table>
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>rwo</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 long … so 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
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 Modification

• 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
    \texttt{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 so honors principle of fail-safe defaults
Handling 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 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
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) \}
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
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 (con’t)

- 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 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
Limits

• 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!
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

• 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
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 key is lock; deciphering key is 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 $o' = (E_1(o), \ldots, E_n(o))$
  - Requires consent of all $n$ to access $o$: store $o' = (E_1(E_2(\ldots(E_n(o))\ldots)))$
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 \n    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 …
More Examples

• LOCK system:
  – Compiler produces “data”
  – Trusted process must change this type to “executable” before program can be executed

• Sidewinder firewall
  – Subjects assigned domain, objects assigned type
    • Example: ingress packets get one type, egress packets another
  – All actions controlled by type, so ingress packets cannot masquerade as egress packets (and vice versa)
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
PACLs

• Propagated Access Control List
  – Implements ORGON

• 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: $PACL_s$ means $s$ created object; $PACL(e)$ is PACL associated with entity $e$
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 *dc*
  \[
  \text{PACL}(dc) = \text{PACL}_{\text{Betty}} \cap \text{PACL}_{\text{Ann}}
  \]

- PACL_{\text{Betty}} allows Char to access objects, but PACL_{\text{Ann}} does not; both allow June to access objects
  - June can read *dc*
  - Char cannot read *dc*