Lecture 13
October 25, 2023

CAs and Policies

- Matt Bishop wants a certificate from Certs-from-Us
  - How does Certs-from-Us know this is "Matt Bishop"?
    - CA's *authentication policy* says what type and strength of authentication is needed to identify Matt Bishop to satisfy the CA that this is, in fact, Matt Bishop
  - Will Certs-from-Us issue this "Matt Bishop" a certificate once he is suitably authenticated?
    - CA's *issuance policy* says to which principals the CA will issue certificates

Registration Authority

- Third party delegated by CA the authority to check data to be put into certificate
  - This includes identity
- RA determines whether CA's requirements are met
- If so, then it informs CA to issue certificates

Internet Certification Hierarchy

- Tree structured arrangement of CAs
  - Root is *Internet Policy Registration Authority*, or IPRA
    - Sets policies all subordinate CAs must follow
    - Certifies subordinate CAs (called *policy certification authorities*, or PCAs), each of which has own authentication, issuance policies
    - Does not issue certificates to individuals or organizations other than subordinate CAs
  - PCAs issue certificates to ordinary CAs
    - Does not issue certificates to individuals or organizations other than subordinate CAs
  - CAs issue certificates to organizations or individuals

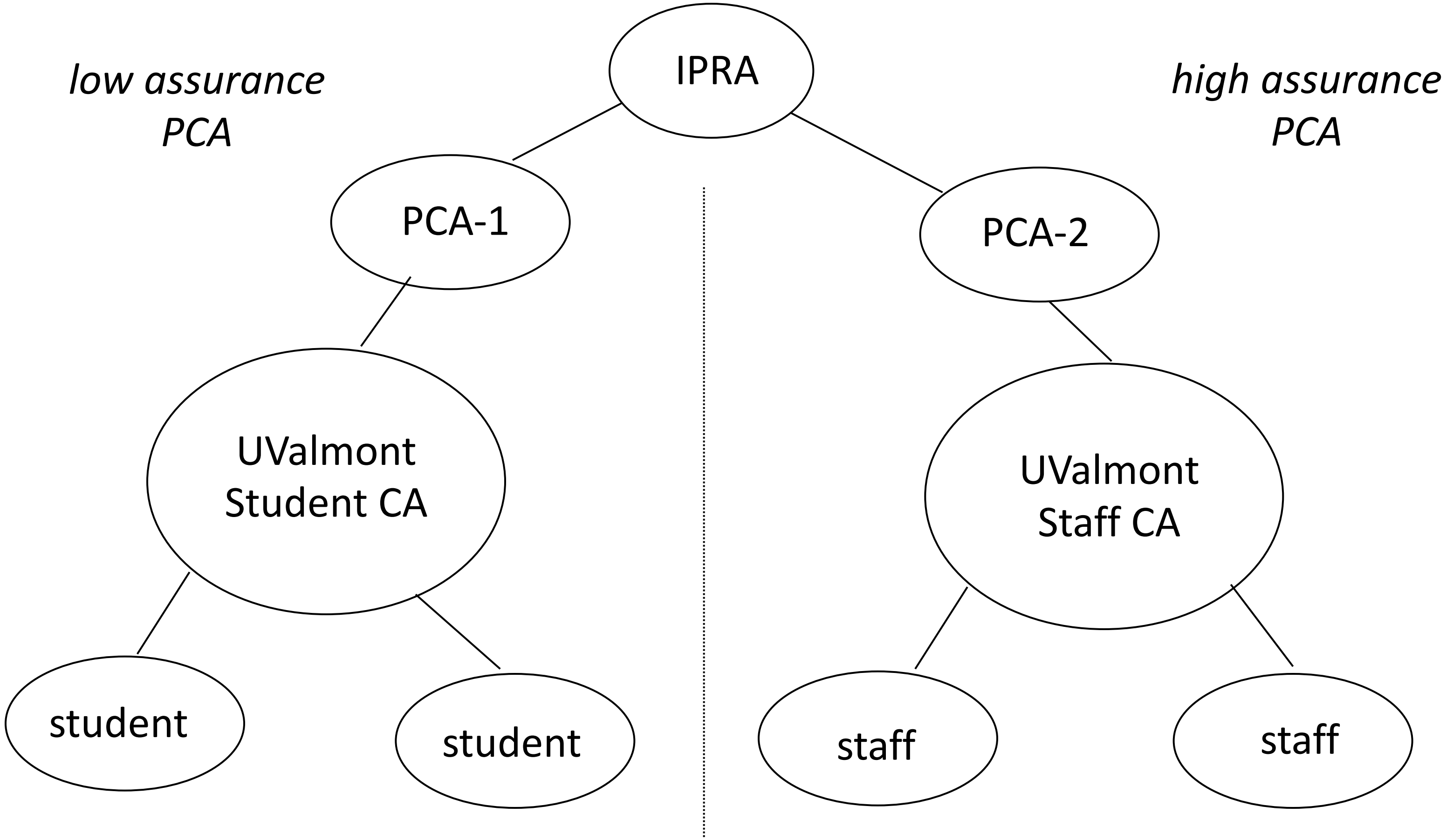
Example

- University of Valmont issues certificates to students, staff
  - Students must present valid reg cards (considered low assurance)
  - Staff must present proof of employment and fingerprints, which are compared to those taken when staff member hired (considered high assurance)

UValmont and PCAs

- First PCA: requires subordinate CAs to make good-faith effort to verify identities of principals to whom it issues certificates
  - Student authentication requirements meet this
- Second PCA: requires use of biometrics to verify identity
  - Student authentication requirements do not meet this
  - Staff authentication requirements do meet this
- UValmont establishes to CAs, one under each PCA above

UValmont and Certification Hierarchy



Certificate Differences

- Student, staff certificates signed using different private keys (for different CAs)
  - Student’s signed by key corresponding to low assurance certificate signed by first PCA
  - Staff’s signed by key corresponding to high assurance certificate signed by second PCA
- To see what policy used to authenticate:
  - Determine CA signing certificate, check its policy
  - Also go to PCA that signed CA’s certificate
    - CAs are restricted by PCA’s policy, but CA can restrict itself further

Types of Certificates

- Organizational certificate
  - Issued based on principal's affiliation with organization
  - Example Distinguished Name

    /O=University of Valmont/OU=Computer Science Department/CN=Marsha Merteuille/
- Residential certificate
  - Issued based on where principal lives
  - No affiliation with organization implied
  - Example Distinguished Name

    /C=US/SP=Louisiana/L=Valmont/PA=1 Express Way/CN=Marsha Merteuille/

Certificates for Roles

- Certificate tied to a role
- Example
  - UValmont wants comptroller to have a certificate
    - This way, she can sign contracts and documents digitally
  - Distinguished Name

    /O=University of Valmont/OU=Office of the Big Bucks/RN=Comptroller/

    where “RN” is *role name*; note the individual using the certificate is not named, so no CN

Certificate Principal Identifiers

- Need not be Distinguished Names
  - Example: PGP certificates usually have email addresses, not Distinguished Names
- Permits ambiguity, so the user of the certificate may not be sure to whom it refers
  - Email addresses change often, particularly if work email addresses used
- Problem: how do you prevent naming conflicts?

Naming Conflicts

- X.509, PGP silent
  - Assume CAs will prevent name conflicts as follows
    - No two distinct CAs have the same Distinguished Name
    - No two principals have certificates issued containing the same Distinguished Name by a single CA

Internet Certification Hierarchy

- In theory, no naming collisions
  - IPRA requires each PCA to have a unique Distinguished Name
  - No PCA may certify two distinct CAs with same Distinguished Name
- In practice, considerable confusion possible!

Example Collision

John Smith, John Smith Jr. live at same address

- John Smith Jr. applies for residential certificate from Certs-from-Us, getting the DN of:
  /C=US/SP=Maine/L=Portland/PA=1 First Ave./CN=John Smith/
- Now his father applies for residential certificate from Quick-Certs, getting DN of:
  /C=US/SP=Maine/L=Portland/PA=1 First Ave./CN=John Smith/
  because Quick-Certs has no way of knowing that DN is taken

Solutions

- Organizational certificates
  - All CA DNs must be superior to that of the principal
  - Example: for Marsha Merteuille's DN:

    /O=University of Valmont/OU=Computer Science Department/CN=Marsha Merteuille/

    DN of the CA must be either:

    /O=University of Valmont/

    (the issuer being the University) or

    /O=University of Valmont/OU=Computer Science Department/

    (the issuer being the Department)

Solutions

- Residential certificates
  - DN collisions explicitly allowed (in above example, no way to force disambiguation)

    /C=US/SP=Maine/L=Portland/PA=1 First Ave./CN=John Smith/

    Unless names of individuals are different, how can you force different names in the certificates?

Related Problem

- Single CA issues two types of certificates under two different PCAs
- Example
  - UValmont issues both low assurance, high assurance certificates under two different PCAs
  - How does validator know under which PCA the certificate was issued?
    - Reflects on assurance of the identity of the principal to whom certificate was issued

Solution

- CA Distinguished Names need *not* be unique
- CA (Distinguished Name, public key) pair *must* be unique
- Example
  - In earlier UValmont example, student validation required using first PCA's public key; validation using second PCA's public key would fail
  - Keys used to sign certificate indicate the PCA, and the policy, under which certificate is issued

Meaning of Identity

- Authentication validates identity
  - CA specifies type of authentication
  - If incorrect, CA may misidentify entity unintentionally
- Certificate binds *external* identity to crypto key and Distinguished Name
  - Need confidentiality, integrity, anonymity
    - Recipient knows same entity sent all messages, but *not* who that entity is

Persona Certificate

- Certificate with meaningless Distinguished Name
  - If DN is
    /C=US/O=Microsoft Corp./CN=Bill Gates/
    the real subject may not (or may) be Mr. Gates
  - Issued by CAs with persona policies under a PCA with policy that supports this
- PGP certificates can use any name, so provide this implicitly

Example

- Government requires all citizens with gene X to register
  - Anecdotal evidence people with this gene become criminals with probability 0.5.
  - Law to be made quietly, as no scientific evidence supports this, and government wants no civil rights fuss
- Government employee wants to alert media
  - Government will deny plan, change approach
  - Government employee will be fired, prosecuted
- Must notify media anonymously

Example

- Employee gets persona certificate, sends copy of plan to media
  - Media knows message unchanged during transit, but not who sent it
  - Government denies plan, changes it
- Employee sends copy of new plan signed using same certificate
  - Media can tell it’s from original whistleblower
  - Media cannot track back whom that whistleblower is

Trust

- Goal of certificate: bind correct identity to DN
- Question: what is degree of assurance?
- X.509v4, certificate hierarchy
  - Depends on policy of CA issuing certificate
  - Depends on how well CA follows that policy
  - Depends on how easy the required authentication can be spoofed
- Really, estimate based on the above factors

Example: Passport Required

- DN has name on passport, number and issuer of passport
- What are points of trust?
  - Passport not forged and name on it not altered
  - Passport issued to person named in passport
  - Person presenting passport is person to whom it was issued
  - CA has checked passport and individual using passport

PGP Certificates

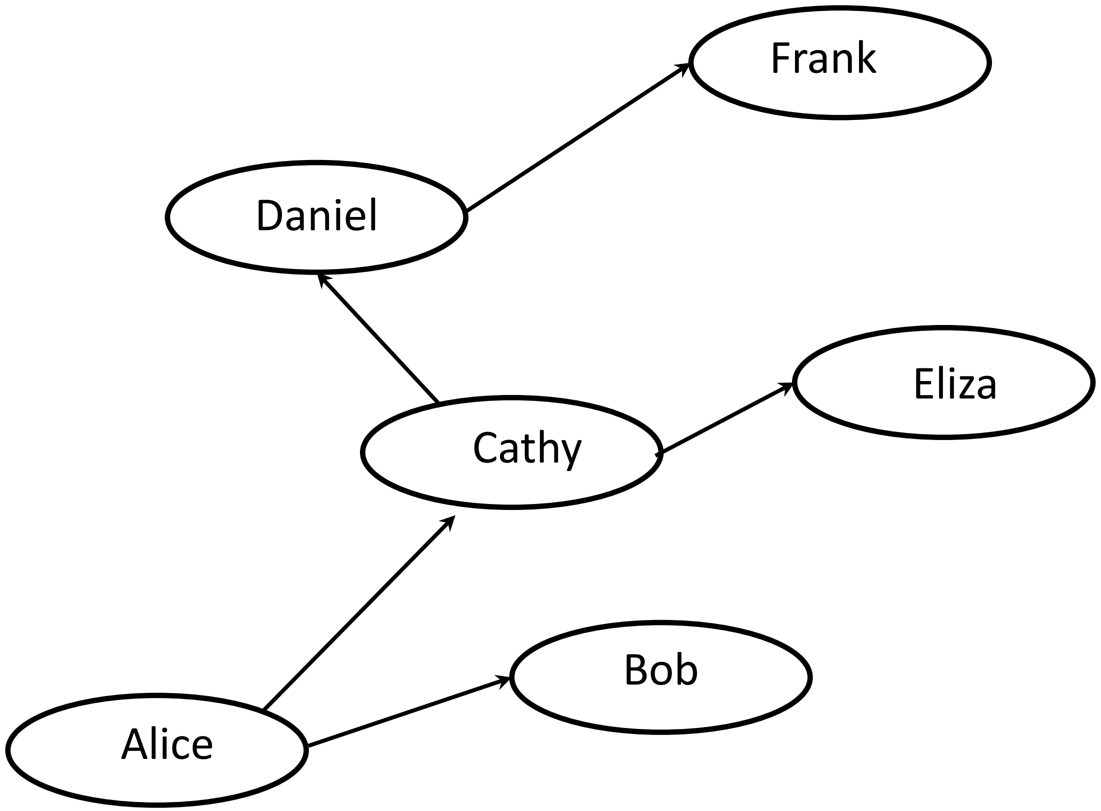
- Public key packet
  - Version
  - Time of creation
  - Validity period
  - Public key algorithm and parameters
  - Public key
- Followed by 0 or more signature packets

- Signature packet (OpenPGP v3)
  - Version
  - Signature type (trust level)
  - Creation time
  - Key identifier of the signer
  - Public key algorithm
  - Hash algorithm
  - Part of signed hash value
  - Signature

PGP Certificates

- Level of trust in signature field signature type
- Four levels
  - Generic (no trust assertions made)
  - Persona (no verification)
  - Casual (some verification)
  - Positive (substantial verification)
- What do these mean?
  - Meaning not given by OpenPGP standard
  - Signer determines what level to use
  - Casual to one signer may be positive to another

Web of Trust



Alice needs Frank's certificate

- She doesn't have it so she asks Bob and Cathy if they do
- Neither do, so Cathy asks Daniel and Eliza
- Daniel knows Frank and gets his public key
- Daniel decides how much he trusts Frank and that the certificate is Frank's, and forwards both to Cathy
- Daniel decides how much he trusts Frank and that the certificate is Frank's, and forwards both to Cathy
- Cathy decides how much she trusts Daniel, and forwards that and the certificate to Alice
- Alice decides whether to accept the certificate as legitimate or reject it.

Note: no certification or registration authorities needed

Access Control Mechanisms

- Access control lists
- Capability lists
- Ring-based access control

Access Control Lists

- Columns of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

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
    - Most UNIX-like systems: if directory has setgid permission, file group owned by group of directory (Solaris, Linux)

ACLs + Abbreviations

- Augment abbreviated lists with ACLs
  - Intent is to shorten ACL
- ACLs override abbreviations
  - Exact method varies
- Example: Extended permissions (Linux, FreeBSD, others)
  - Minimal ACLs are abbreviations, extended ACLs give specific users, groups permissions
  - Extended ACL entries give rights provided those rights are in mask

Minimal and Extended ACL

user *heidi*, group *family* owns file with permissions:

```
user::rw-
user:skyler:rwx
group::rw-
group:child:r--
mask::rw-
other::r--
```

- *heidi* can read, write file (first line)
- *matt*, not in group *child*, can read file (last line)
- *skyler* can read, write file (second line masked by fifth line)
- *sage*, in group *family*, can read, write the file (third line masked by fifth line)
- *steven*, in group *child*, can read file (fourth line masked by fifth line)

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
- 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 …

Capability Lists

- Columns of access control matrix

	file1	*file2*	*file3*
Andy	rx	r	rwo
Betty	rwxo	r	
Charlie	rx	rwo	w

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
    - EROS does 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

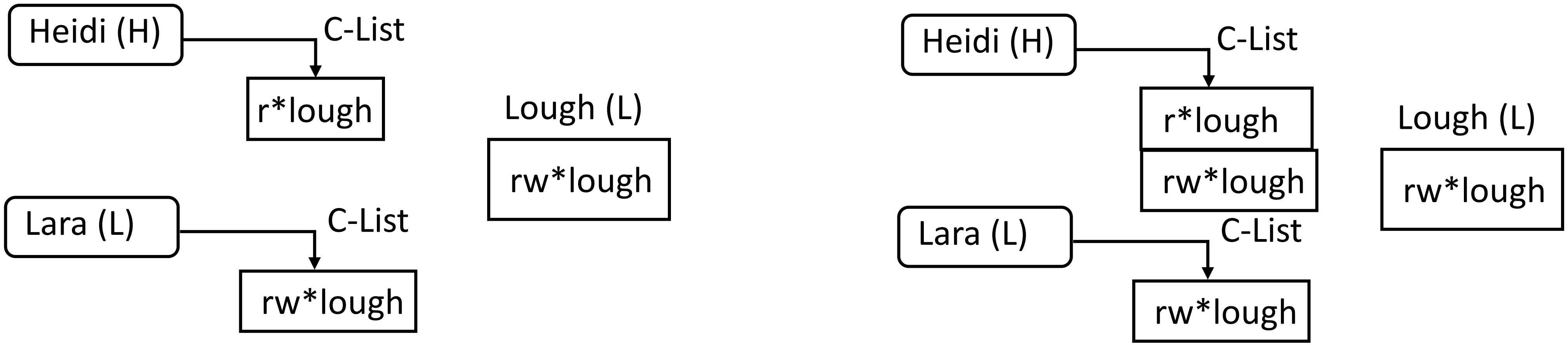
- HYDRA: templates
  - Associated with each procedure, function in module
  - Adds rights to process capability *while the procedure or function is being executed*
  - Rights deleted on exit
- Intel iAPX 432: access descriptors for objects
  - These are really capabilities
  - 1 bit in this controls 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 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” and “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

Privileges

- In Linux, used to override or add access restrictions by adding, masking rights
  - Not capabilities as no particular object associated with the (added or deleted) rights
- 3 sets of privileges
  - Bounding set (all privileges process may assert)
  - Effective set (current privileges process may assert)
  - Saved set (rights saved for future purpose)
- Example: UNIX effective, saved UID

Trusted Solaris

- Associated with each executable:
  - *Allowed set* (*AS*) are privileges assigned to process created by executing file
  - *Forced set* (*FS*) are privileges process must have when it begins execution
  - $FS \subseteq AS$

Trusted Solaris Privileges

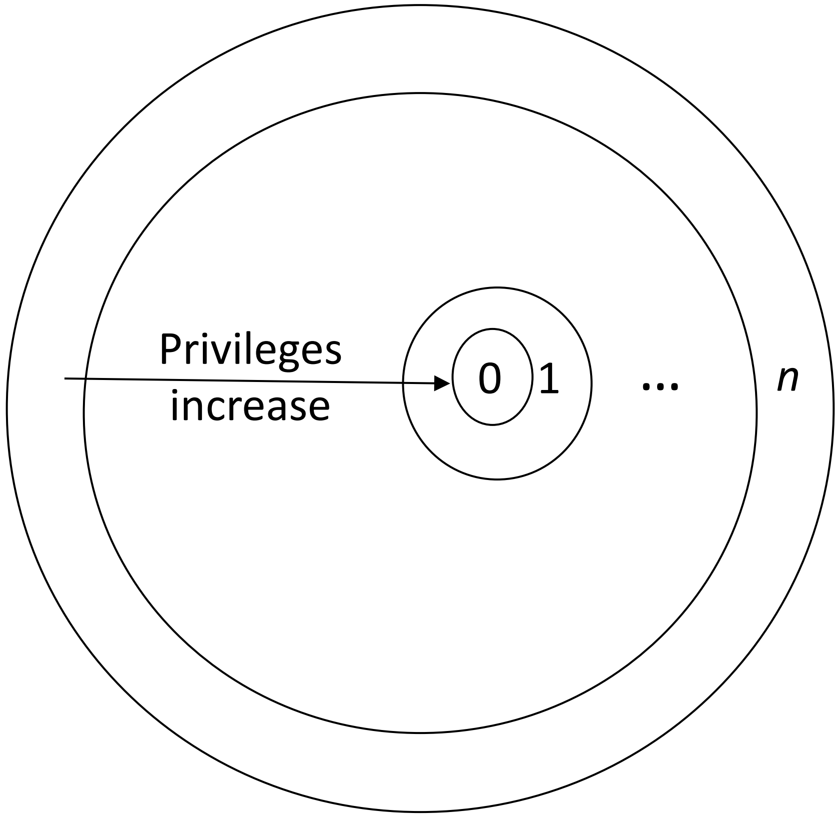
Four sets:

- *Inheritable set* (*IS*): privileges inherited from parent process
- *Permitted set* (*PS*): all privileges process may assert; $(FS \cup IS) \cap AS$
  - Corresponds to bounding set
- *Effective set* (*ES*): privileges program requires for current task; initially, *PS*
- *Saved set* (*SS*): privileges inherited from parent process and allowed for use; that is, $IS \cap AS$

Bracketing Effective Privileges

- Process needs to read file at particular point
- *file_mac_read, file_dac_read* ∈ *PS*, *ES*
- Initially, program deletes these from *ES*
  - So they can’t be used
- Just before reading file, add them back to *ES*
  - Allowed as these are in *PS*
- When file is read, delete from *ES*
  - And if no more reading, can delete from *PS*

Ring-Based Access Control



- Process (segment) accesses another segment
  - read (data)
  - execute (routine)
- *Gate* is an entry point for calling segment
- Rights:
  - *r* read
  - *w* write
  - *a* append
  - *e* execute