CIS551: Computer and Network Security

Jonathan M. Smith

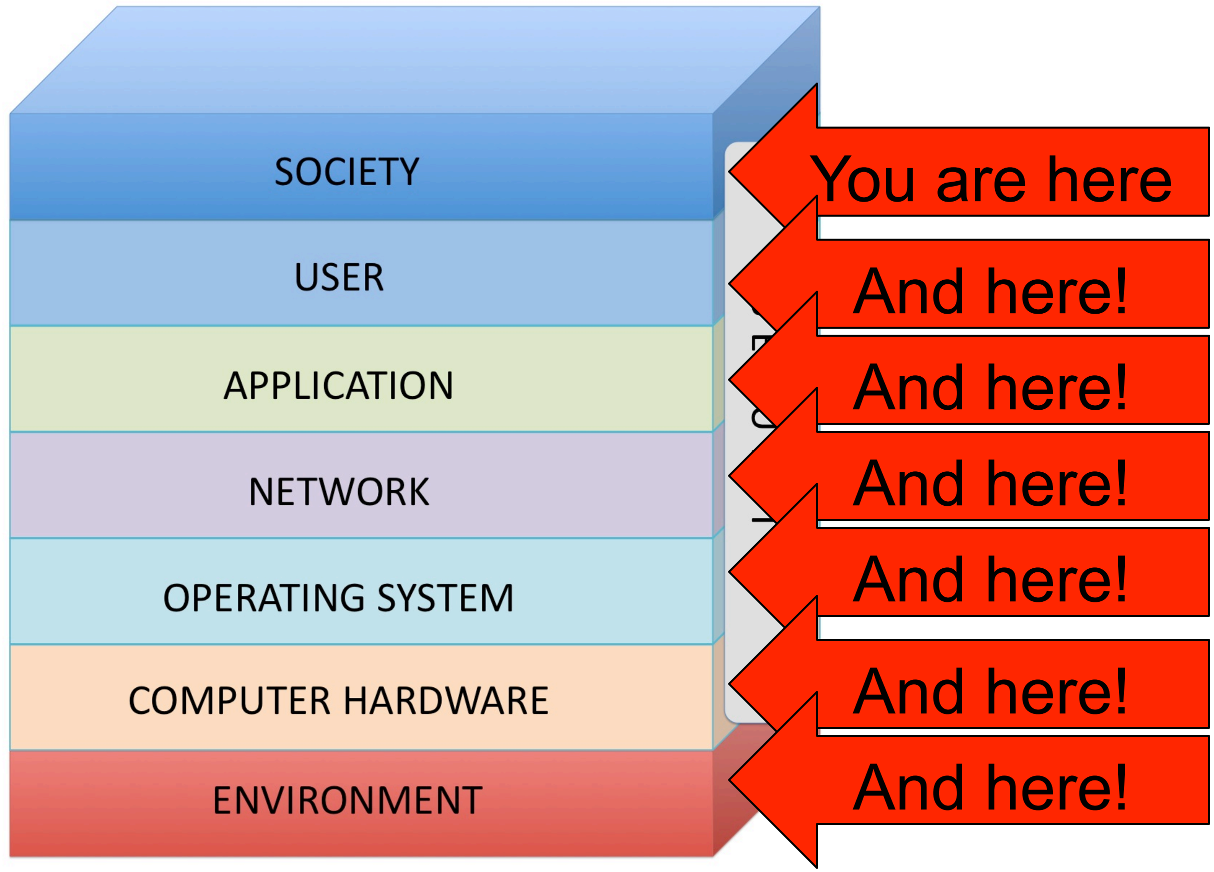
jms@cis.upenn.edu

01/29/2014

CIS551 Topics

- Computer Security
  - Software/Languages, Computer Arch.
  - Access Control, Operating Systems
  - Threats: Vulnerabilities, Viruses
- Computer Networks
  - Physical layers, Internet, WWW, Applications
  - Cryptography in several forms
  - Threats: Confidentiality, Integrity, Availability
- Systems Viewpoint
  - Users, social engineering, insider threats

Sincoskie NIS model



W.D. Sincoskie, *et al.* "Layer Dissonance and Closure in Networked Information Security" (white paper)

Access Control

- Security is doing the right thing for the right person at the right place at the right time – nothing more or less
  - **Subject**: actor (e.g., process, user, host)
  - **Object**: acted on (e.g., file)
  - **Action**: Operation on **Object** by **Subject**
  - **Right**: Permission for Action

Access Control: Examples

- Assume OS is a **Subject** with all **Right**s
- To create a file *f* owned by Alice:
  - Create **Object** *f*
  - Grant ‘own’ to Alice with respect to *f*
  - Grant ‘read’ to Alice with respect to *f*
  - Grant ‘write’ to Alice with respect to *f*
- To start a login for Alice
  - Input and check password
  - Create a shell process *p*
  - Grant ‘own_process’ to Alice with respect to *p*

Implementing Access Control

- Access control matrices
  - **#Subject**s >> #users (say 1000s per user)
  - **#Object**s >> **#Subject**s (say 1,000,000s)
  - To specify "all users read *f*"
    - Change O(users) entries
- Matrix is typically sparse
  - Store only non-empty entries
- Special consideration for *groups* of users

Access Control Matrices

A[s][o]	Obj_1	Obj_2	…	Obj_N
$Subj_1$	{r,w,x}	{r,w}	…	{}
$Subj_2$	{w,x}	{}	…	
…	…	…	…	
$Subj_M$	{x}	{r,w,x}	…	{r,w,x}

Each entry contains a set of **Right**s.

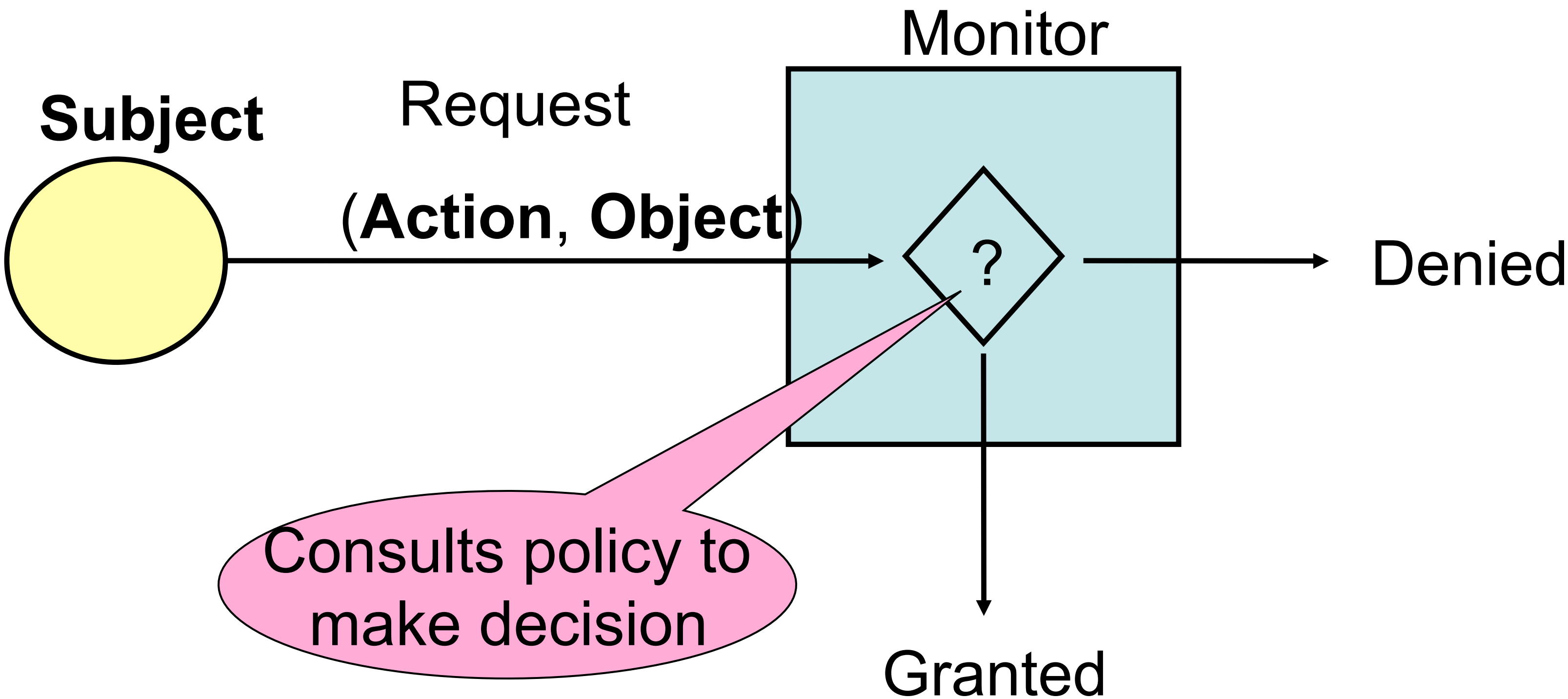
Rights

- Besides read, write, execute **Right**s there are many others:
  - Ownership
  - Creation
    - New **Subject**s (*i.e*., in *n*x add a *user*)
    - New **Object**s (*i.e.*, create a new *file*)
    - New **Right**s: Grant **Right** *r* to **Subject** *s* with respect to **Object** *o* (sometimes called delegation)
  - Deletion of
    - **Subject**s
    - **Object**s
    - **Right**s (sometimes called *revocation*)

Access Control Checks

- Suppose **Subject** *s* wants to perform **Action** that requires **Right** *r* on **Object** *o*:
- If ($r \in A[s][o]$) then <u>*perform action*</u> else <u>*access is denied*</u>
  - In *n*x, this is done via `namei()`

Reference Monitors



Reference Monitors

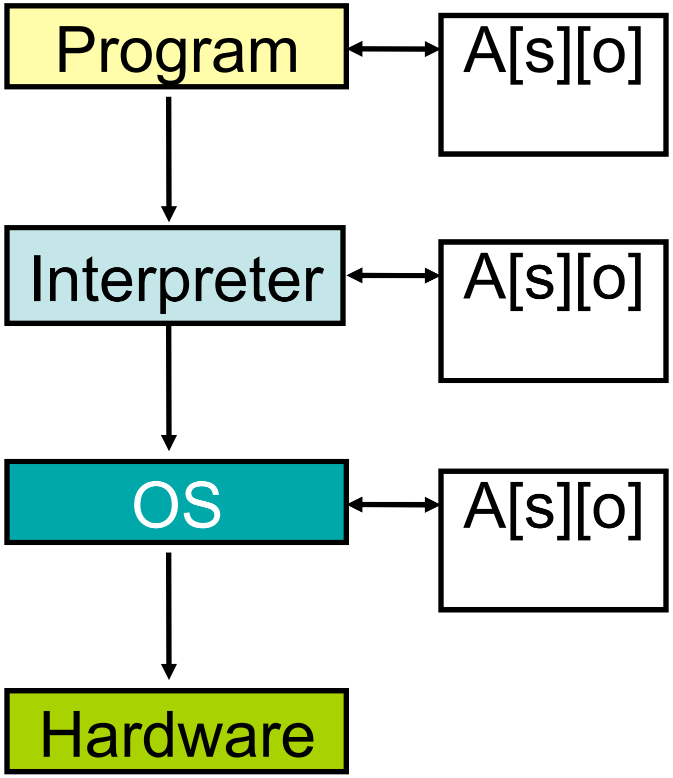
- Criteria
  - Correctness
  - Complete mediation (all avenues of access must be protected)
  - Expressiveness (what policies are admitted)
  - How large/complex is the mechanism?
- Trusted Computing Base (TCB)
  - The set of components that must be trusted to enforce a given security policy
  - Would like to simplify/minimize the TCB to improve assurance of correctness

Protecting Reference Monitors

- It must not be possible to circumvent the reference monitor by corrupting it
- Protection Mechanisms
  - Type checking
  - Sandboxing: run processes in isolation
  - Software fault isolation: rewrite memory access instructions to perform bounds checking
  - User/Kernel modes
  - Segmentation of memory (OS resources aren't part of process virtual memory system)
  - Physical configuration (e.g., network topology)

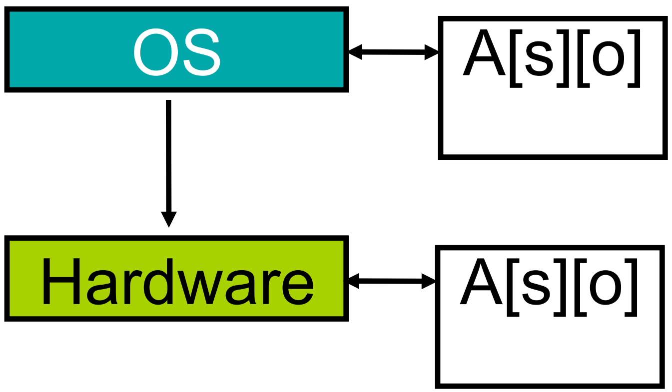
Software Mechanisms

- Interpreters
  - Check the execution of every instruction
  - Hard to mediate high-level abstractions
- Wrappers
  - Only “interpret” some of the instructions
  - What do you wrap?
  - Where do you wrap? (link-time?)
- Operating Systems
  - Level of granularity?
  - Context switching overheads?
- Example
  - Java and C# runtime systems



Hardware Mechanisms

- Multiple modes of operation
  - User mode (problem state)
  - Kernel mode (supervisor state)
- Specialized hardware
  - Virtual memory support (TLB's, etc.)
  - Interrupts



Access Control Lists

A[s][o]	Obj_1	Obj_2	…	Obj_N
$Subj_1$	{r,w,x}	{r,w}	…	{}
$Subj_2$	{w,x}	{}	…	{r}
…	…	…	…	…
$Subj_M$	{x}	{r,w,x}	…	{r,w,x}

For each **Object**, store a *list of (***Subject x Rights****) pairs*.

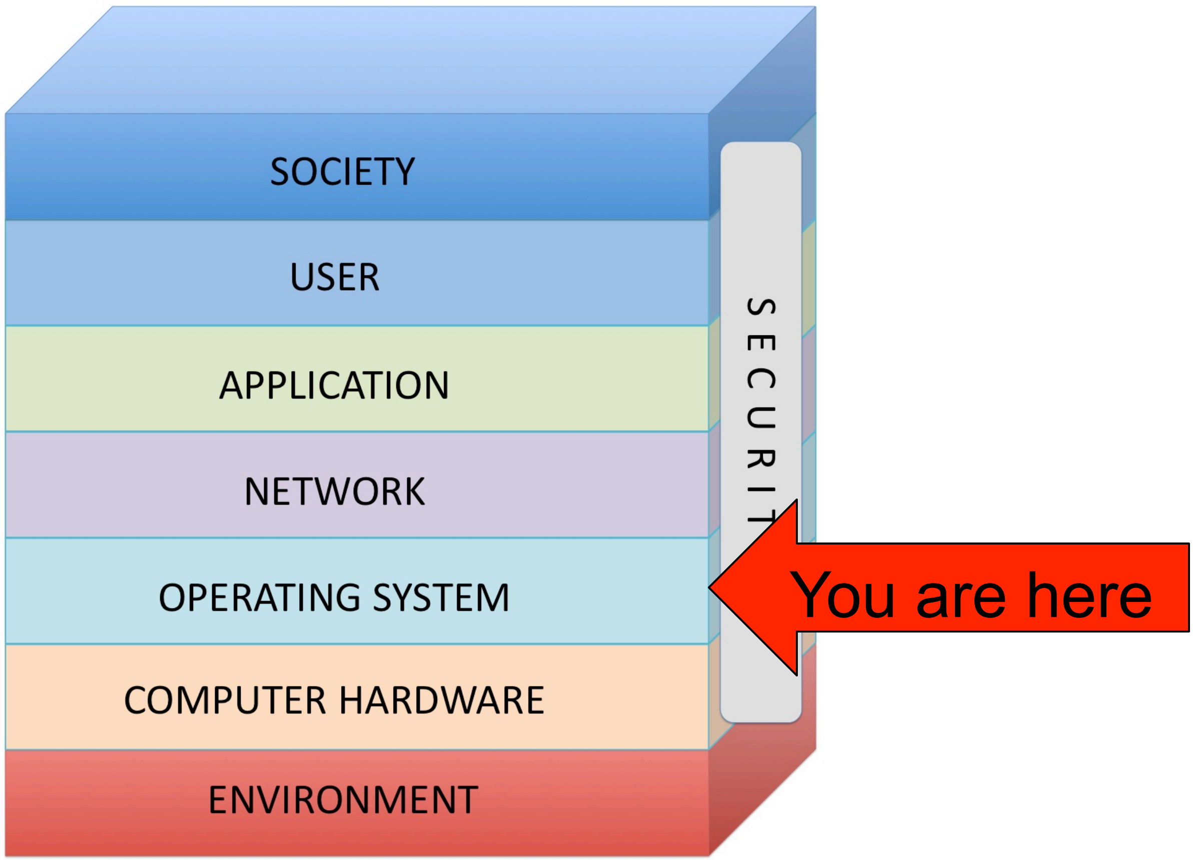
Access Control Lists

- Resolving queries is linear in length of the list
- Revocation w.r.t. a single **Object** is easy
- "*Who can access this object?*" is easy
  - Useful for auditing
- Lists could be long
  - Factor into groups (lists of **Subject**s)
  - Give permissions based on group
- Authentication critical
  - When does it take place? Every access would be expensive (`namei()` -> `fd`).

Representational Completeness

- Access Control Lists
  - Can represent any access control matrix
  - Potentially very large
  - Used in Windows file system, NTFS
- Unix file permissions (next topic)
  - Fixed size
  - Cannot naturally express some access control policies/matrices

Sincoskie NIS model



W.D. Sincoskie, *et al.* “Layer Dissonance and Closure in Networked Information Security” (white paper)

*n*x file security

- Each file has owner and group
- Permissions set by owner
  - Read, write, execute
  - Owner, group, other
  - Represented by vector of four octal values

setid

$\text{---}\ \underbrace{\text{rwx}}_{\text{owner}}\ \underbrace{\text{rwx}}_{\text{group}}\ \underbrace{\text{rwx}}_{\text{other}}$

- Only owner + root can change permissions
  - This privilege cannot be delegated or shared
- Setid bits (type “`man setuid`” for details)

Question

- "owner" can have fewer privileges than "other"
  - What happens?
    - User gets access?
    - User does not?

- Prioritized resolution of differences

  if user = owner then *owner* permission

  else if user in group then *group* permission

  else *other* permission

Setid bits on executable *n*x file

- Three setid bits
  - Sticky
    - Off: if user has write permission on directory, can rename or remove files, even if not owner
    - On: only file owner, directory owner, and root can rename or remove file in the directory
  - Setuid – set EUID of process to ID of file owner
    - passwd owned by root and setuid is true
    - Jeff executes passwd: “passwd runs as root”
  - Setgid – set EGID of process to GID of file

*n*x Policies Interact

```
drwx------  129 jms  jms  4454 Mar 16 10:12 /Users/jms/
-rw-r--r--    1 jms  jms   148 Jan 20  2008 /Users/jms/.profile
```

- *stevez* cannot read `/Users/jms/.profile`
  - The confidentiality/availability of an object depends on policies other than its own!
  - Such interactions make specifying policies hard.
  - Problem is not limited to *n*x (or file systems).

*n*x summary

- We're all very used to this …
  - So probably seems pretty good
  - We overlook ways it might be better
- Good things
  - Some protection from most users
  - Flexible enough to make things possible
- Main bad thing
  - Too tempting to use root privileges
  - No way to assume some root privileges without all root privileges

Capabilities

- A *capability* is a (**Object**, **Rights**) pair
  - Used like a movie ticket, *e.g.*:
    ("GreenZone", {admit one, 7:00pm show})
- Should be *unforgeable*
  - Otherwise, **Subjects** could get illegal access
- Authentication takes place when the capabilities are granted (not needed at use)
- Harder to do revocation (must find *all* tickets)
- Easy to audit a **Subject**, hard to audit an **Object**

Capabilities Lists

A[s][o]	Obj_1	Obj_2	…	Obj_N
$Subj_1$	{r,w,x}	{r,w}	…	{}
$Subj_2$	{w,x}	{}	…	{r}
…	…	…	…	…
$Subj_M$	{x}	{r,w,x}	…	{r,w,x}

For each **Subject**, store a *list of (***Object x Rights***) pairs*.

Implementing Capabilities

- Must be able to name **Object**s
- Unique identifiers (UIDs)
  - Must keep map of UIDs to **Object**s
  - Must protect integrity of the map
  - Extra level of indirection to use the **Object**
  - Generating UIDs can be difficult
- Pointers
  - Name changes when the **Object** moves
  - Remote pointers in distributed setting

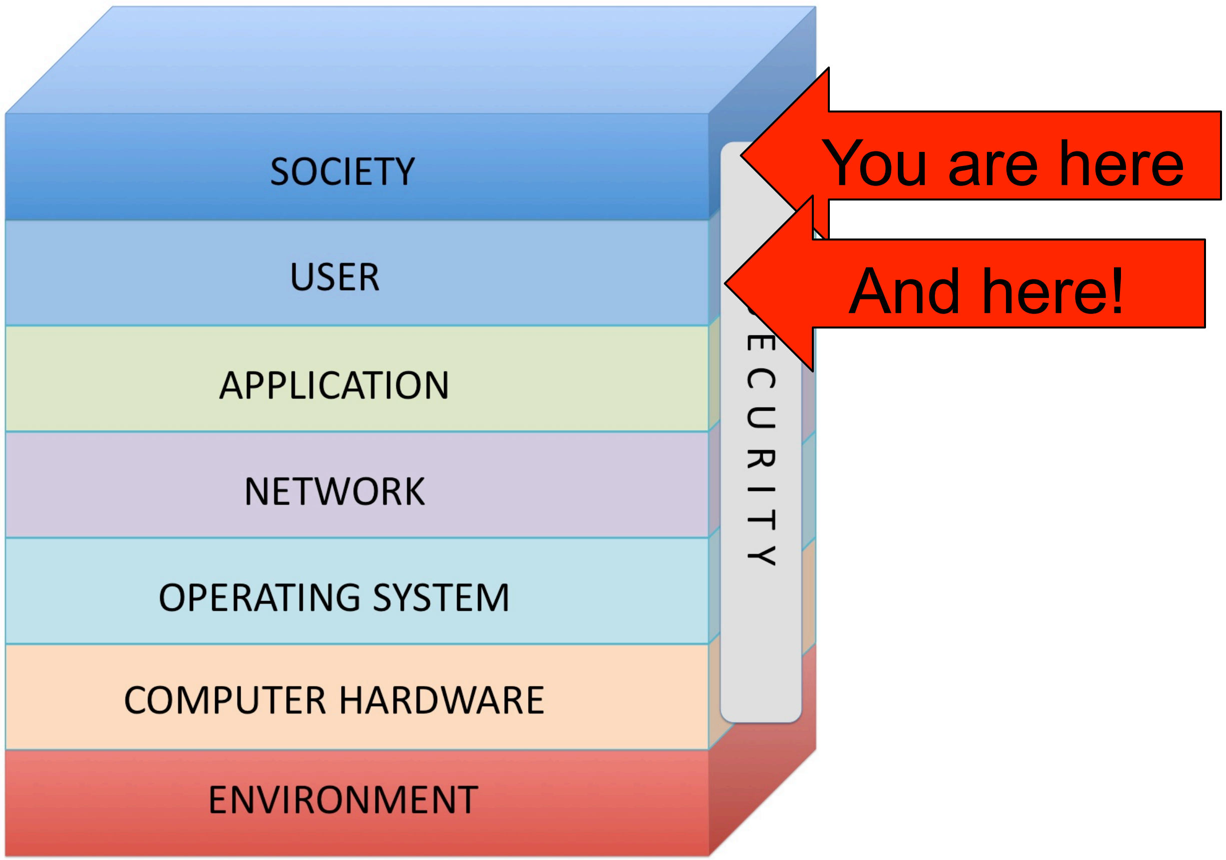
Unforgeability of Capabilities

- Special hardware: tagged words in memory
  - Can't copy/modify tagged words
  - Example: Intel 432
- Store the capabilities in protected address space (e.g., EROS)
- Use cryptographic techniques
  - OS kernel could sign (**Object, Rights**) pairs using a private key
  - Any process can verify the capability

Multilevel Security

- Multiple levels of confidentiality or integrity ratings
- Group individuals and resources
  – Use some form of hierarchy to organize policy
- Trivial example: Public $\leq$ Secret
- *Information flow*
  – Regulate how information is used throughout entire system
  – A document generated from both Public and Secret information must be rated Secret.
  – Intuition: "Secret" information should not flow to "Public" locations.

Sincoskie NIS model



W.D. Sincoskie, *et al.* “Layer Dissonance and Closure in Networked Information Security” (white paper)

Military security policy

- Classification - multiple levels of sensitivity
  – Notions of classification and clearance
- Do not let classified information “leak”

Top Secret
Secret
Confidential
Restricted
Unclassified