INF523: Computer System Assurance

Formal Methods (continued)

Prof. Clifford Neuman

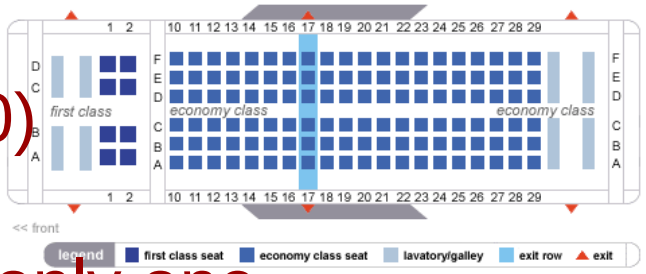
Lecture 10
30 October 2020

Formal System Specifications

- Previous example used first-order logic (FOL)
  - ∀ and ∃
- For complex systems, FOL may not be enough
- Want "higher-order" logic (HOL), which can take functions as arguments
- E.g., [Rushby PVS phone book example]
  - http://www.csl.sri.com/papers/wift-tutorial/slides.pdf

Homework (due 11/6/2020)

- Write a formal spec for seating in an airplane:



- An airplane has 100 seats (1..100)
- Every passenger gets one seat
- Any seat with a passenger holds only one passenger
- The state of a plane P is a function [N -> S]
  - Maps a passenger name to a seat number
- Two functions: assign_seat and deassign_seat
- Define the functions
- Show some lemmas that demonstrate correctness

Start of Homework Solution

- Types:
  - N : type (of passenger)
  - S : type (of seat number)
  - A : type (of airplane function) [N -> S]
  - e0 : N (represents an empty seat)

- Variables:
  - nm : var N (a passenger)
  - pl : var A (an airplane function)
  - st : var S (a seat number)

What you Need to Do

1. Define the axioms for the two functions:
   - assign_seat : [A x N x S -> A]
   - deassign_seat : [A x S -> A]
2. Be careful that the spec covers all requirements:
   - Can someone have “e0” as their seat number?
   - Can a passenger have more than one seat?
   - Can a seat have more than one passenger?
3. Identify some lemmas that demonstrate that the system specification describes what is intended and sketch the proof

INF523: Assurance in Cyberspace as Applied to Information Security

Case Studies of Formal Specification and Proofs

Extra Session Tuesday 16 November

- Process Control and Medical Devices (40 min)
  - Medical Devices – Jaynee Shah
  - Industrial Control Systems - Venkat Ramana Reddy Mareddy
- The Cloud and Storage/Database Infrastructure (80 min)
  - Database Servers – Di Rama
  - Cloud Security - Shagun Bhatia
  - FedRamp - Dewaine Reddish
  - Risk Management - Sarahzin Chowdhury
- Isolation and Key Management (40 min)
  - Identity Tokens and Yubikey - Arjun G. Raman
  - Isolation Technologies - Ayush Ambastha

Extra Session Thursday 18 November

Mobile Devices

- Mobile OS - Chinmaya Pandit and Harshit Kothari
- Android - Mohammed Ababtain

Payment

- Apple Pay - Jairo Hernandez
- Apple Pay and Google Pay - MaryLiza Walker
- Apple Pay - Shanice Williams
- Apple Pay - Yang Xue
- Assurance in Payment Systems - Uddipt Sharma

Friday November 20

Operating Systems

- Linux Applications - Aditya Goindi
- Linux - Tejas Pandey
- Chrome OS - Malavika Prabhakar

Infrastructure and Vehicle Control Systems

- US Voting Infrastructure - Anthony Cassar
- Autonomous Vehicles - Chris Samayoa
- Autonomous Vehicles - Amarbir Singh
- Connected and Automated Vehicles - Abhishek Tatti
- Tesla - Dwayne Robinson
- Avionics - Pratyush Prakhar

Formal Verification is Not Enough

- Formal verification complements, but does not replace testing (informal verification)
- Requires abstraction which
  - May leave out important details (stuff missing)
  - May make assumptions that code does not support (extra stuff)
- Even if “proven correct”, may still not be correct

- “Beware of bugs in the above code; I have only proved it correct, not tried it.” -Knuth

Reading for This Class

- Jonathan K. Millen. 1976. Security Kernel validation in practice. *Commun. ACM* 19, 5 (May 1976), 243-250. DOI=10.1145/360051.360059
- T. Levin, S. Padilla, and R. Schell, Engineering Results from the A1 Formal Verification Process, in *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, 1989. pp. 65-74

DEC PDP 11

- Sold by DEC
- 1970s-1990s
- Most popular minicomputer ever
- Smallest mini-computer for a decade that could run Unix

Millen: PDP 11/45 Proof of Correctness

- Proof of correctness for PDP 11/45 security kernel
- Correctness defined as proper implementation of security policy model (BLP)
- Security policy model defined as set of axioms
  - Axioms are propositions from which properties are derived
  - E.g., in BLP, SSC and *-property
- Proof is that all operations available at the interface of the system preserve the axioms
- Also considered covert storage channels
  - Method did not address timing channels

Millen: PDP 11/45 Proof of Correctness

- Security policy model defined as set of axioms
  - Simple security condition
    - If a subject has “read” access to an object, level of subject dominates level of object
  - *-property
    - If a subject has “read” access to one object and “write” access to a second object, level of second object dominates level of first object
  - Tranquility principle for object levels
    - Level of active object will not be changed
  - Exclusion of read access to inactive objects
  - Rewriting (scrubbing) of objects that become active

Layers of Specification and Proof

- Four stages
- Each stage more detailed and closer to machine implementation than the one before

1. FSPM (BLP)
2. FTLS – The interface of the system
   - Includes OS calls and
   - PDP 11/45 instructions available outside kernel
3. Algorithmic specification – High-level code that represents machine language
   - Semantics of language must be well-understood
4. Machine itself: Running code and HW

Why Four Proof Stages?

- Simplify proof work
- Big jump from machine to FSPM
  - FSPM has subjects, objects, *-property, …
  - Machine has code and hardware
- Intermediate layers are closer to each other
- First prove FTLS is valid interpretation of FSPM
- Then further proofs only need to show that lower stages implement FTLS
  - Lower-level proofs don't need abstractions of subjects and objects and *-property

Stages 1 and 2 Specification Format

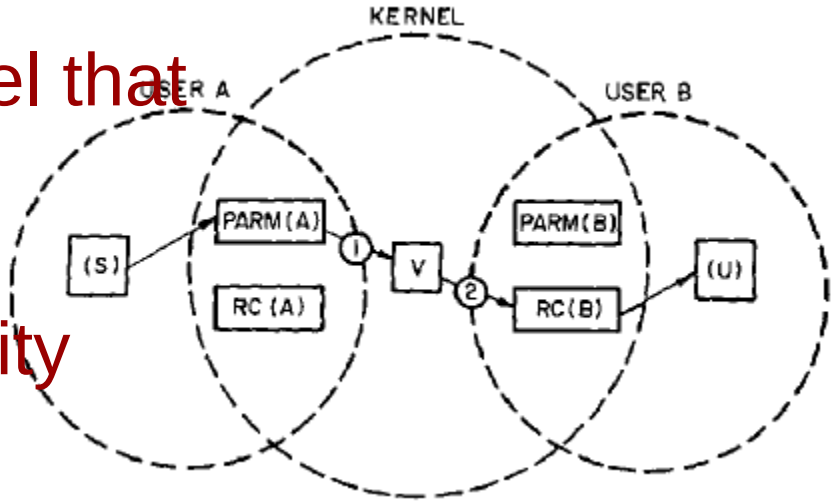
- Both FSPM and FTLS are state machines
  - States and transitions
  - E.g., BLP state is (b, M, f, H)
  - FTLS transitions:
    - Create (activate) object
    - Delete (deactivate) object
    - Get access to an object for a subject
    - Release access to an object for a subject
    - Put a subject in an object's ACL
    - Remove a subject from an object's ACL
    - PDP-11/45 instructions available at interface

V- and O-functions

- State variables and kernel operations are functions
  - State variables are represented as V-functions
    - All V-functions are references to objects
  - Operations are O-functions
    - By subjects to objects
    - Accesses are due to O-function executions
- O-functions have effects on state variables
  - Indicated by values of V-functions before and after
  - E.g., $\neg$`(PS_SEG_INUSE(TCP,dseg)) ∧ RC(TCP) = NO`
  - I.e., if the object is not in use by the subject then the return code is “NO”

"Shared Resource Problems"

- Covert storage (not timing) channels
- User A at high level
- Modifies kernel state variable V
- User B at low level
- Receives value from kernel that was influenced by V
- Detect by assigning security level to internal variables like V



Proof Example in Paper

- Verification that DELETE enforces *-property
- Original spec at right
- Effect statements are labeled “A”, “B”, etc.
  - Used to simplify statement form for proof
- x = ‘y’ means subject has read access to y and write access to x

```
   Function:  DELETE
   parameters:  dseg,entry
   let:  dpn = PS_SEG(TCP,dseg)
   effect:
A    if not PS_SEG_INUSE(TCP,dseg) or
        not AST_WAL(dpn,TCP)  or
B       AST_TYPE(dpn,TCP) ≠ DIRECTORY  or
          ´DIR_SIZE´(dpn,entry) ≠ 0
C    then RC(TCP) = NO
     else RC(TCP) = YES  and
          DIR_SIZE(dpn,entry) = 0  and
          if ´DIR_ACL_HEAD´(dpn,entry) ≠ 0
D         then ACL_CHAIN(dpn,FINDEND(dpn,entry))
                 = ´ACL_CHAIN´(dpn,0)  and
                 ACL_CHAIN(dpn,0) = ´DIR_ACL_HEAD´(dpn,entry)
          end
          and (¥pn)
E(pn)     if pn ≥ dpn.entry
          then if pn = apn.aentry
                  and DIR_TYPE(apn.aentry) = DIRECTORY
               then (¥bentry) DIR_SIZE(pn,bentry) = 0 and
                    (¥acle) ACL_CHAIN(pn,acle)
F(pn)                        = acle + 1 (mod acle_max + 1)
               else VM(pn) = 0
               end  and
               WAIT_SET(pn) = ∅ and
               SMFR_COUNT(pn) = 1
               and (¥aproc)
```

Abbreviated DELETE Specification

- Statements abstracted to show structure
- Bottom version is in "conjunctive form"
- "Else" sometimes replaced by negation of the "If" condition
- Statements in form **`if f then g else h end`** sometimes converted to **`(f ∧ g) ∨ (¬f ∧ h)`**

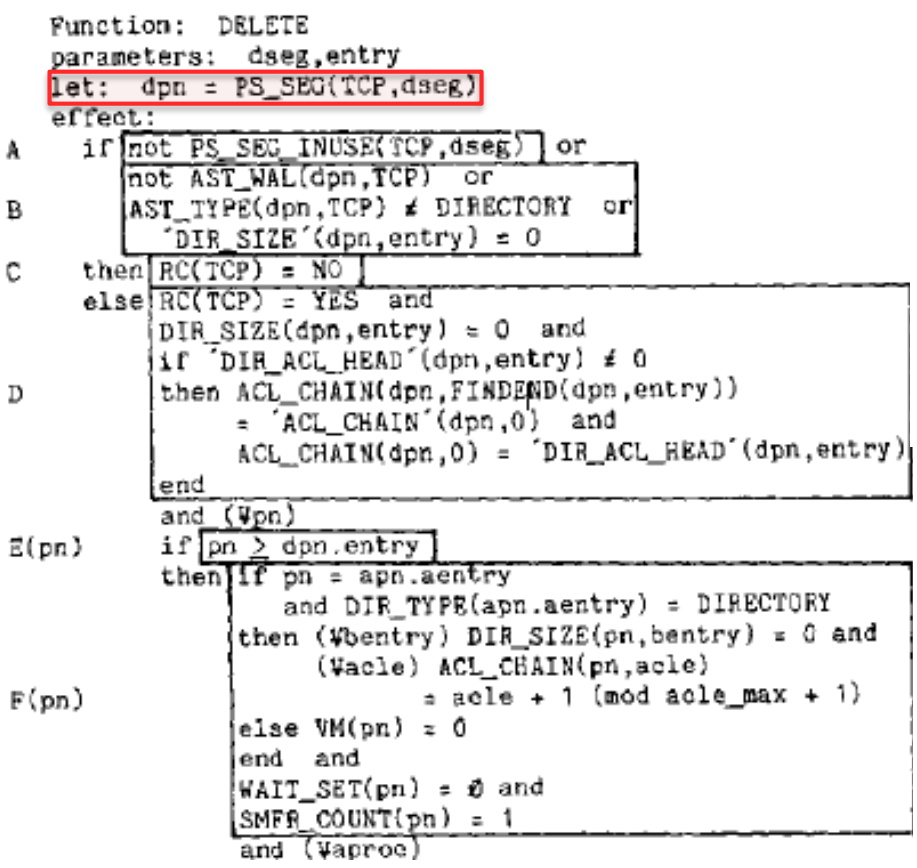
```
If A pr B
then C
else D and
          (¥pn)
          if E(pn)
          then F(pn) and
                    (¥aproc)
                    if G(pn,aproc)
                    then H(pn,aproc)
                    end
          end and
          (¥aproc)
          if J(aproc)
          then K(aproc)
          end
end
```

```
          if A then C else if ~A and B then C else D end
and (¥pn) if ~A and ~B and E(pn) then F(pn) end
and (¥aproc)
          if ~A and ~B
          then (¥pn)
                    if E(pn) and G(pn,aproc)
                    then H(pn,aproc)
                    end
          end
and (¥aproc) if ~A and ~B and J(aproc) then K(aproc) end
```

Proof Technique: Security Levels

- Object levels
  - Level based on pathname *pn* of object in hierarchy
  - Level of object at pathname *pn* is *L(pn)*
  - V-functions that take *pn* as parameter have level *L(pn)*
  - Constant V-functions (no parameters) have sys-low level
- Level of subject with process number *proc* is *PL(proc)*
  - *PL(proc)* is level where subject can both read and write
  - V-functions for subjects (i.e., that read state values for processes, as opposed to system state) have level *PL(proc)*
  - O-functions have process numbers as parameters
  - O-functions and their parameters have level *PL(proc)*

Property Cases and Security Levels



```
   Function:  DELETE
   parameters:  dseg,entry
   let:  dpn = PS_SEG(TCP,dseg)
   effect:
A    if not PS_SEG_INUSE(TCP,dseg) or
B       not AST_WAL(dpn,TCP)  or
        AST_TYPE(dpn,TCP) ≠ DIRECTORY  or
          'DIR_SIZE'(dpn,entry) = 0
C    then RC(TCP) = NO
     else RC(TCP) = YES  and
          DIR_SIZE(dpn,entry) = 0  and
          if 'DIR_ACL_HEAD'(dpn,entry) ≠ 0
D         then ACL_CHAIN(dpn,FINDEND(dpn,entry))
                 = 'ACL_CHAIN'(dpn,0)  and
                 ACL_CHAIN(dpn,0) = 'DIR_ACL_HEAD'(dpn,entry)
          end
          and (¥pn)
E(pn)     if pn ≥ dpn.entry
          then if pn = apn.aentry
                  and DIR_TYPE(apn.aentry) = DIRECTORY
               then (¥bentry) DIR_SIZE(pn,bentry) = 0 and
                    (¥acle) ACL_CHAIN(pn,acle)
F(pn)                     = acle + 1 (mod acle_max + 1)
               else VM(pn) = 0
               end  and
               WAIT_SET(pn) = ∅ and
               SMFR_COUNT(pn) = 1
               and (¥aproc)
```

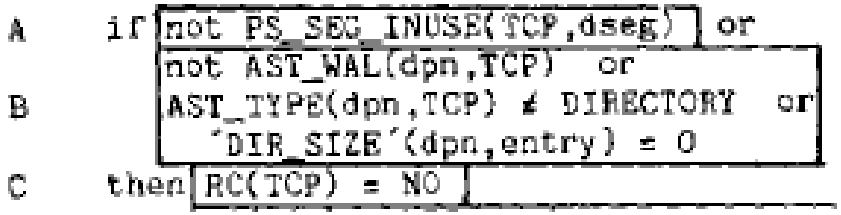
Table I. *-Property Cases.

Subject/ Case	Condi- tions	Ef- fects	Read Levels	Write Levels
S_1/I	A	C	$PL(TCP)$	$PL(TCP)$
S_1/II	$\bar{A}\ B$	C	$PL(TCP), L(dpn)$	$PL(TCP)$
S_1/III	$\bar{A}\ \bar{B}$	D	$PL(TCP), L(dpn)$	$PL(TCP)$, $L(dpn)$
$S_2(pn)$	$\bar{A}\ \bar{B}\ E$	F	$PL(TCP), L(dpn)$, $L(apn)$	$L(pn)$
$S_3(aproc)$	$\bar{A}\ \bar{B}\ E\ G$	H	$PL(TCP), L(dpn)$, $PL(aproc), L(pn)$	$PL(aproc)$
$S_4(aproc)$	$\bar{A}\ \bar{B}\ J$	K	$PL(TCP), L(dpn)$, $PL(aproc)$	$PL(aproc)$

Table II. Security Levels of V-Function References in Labeled Statements.

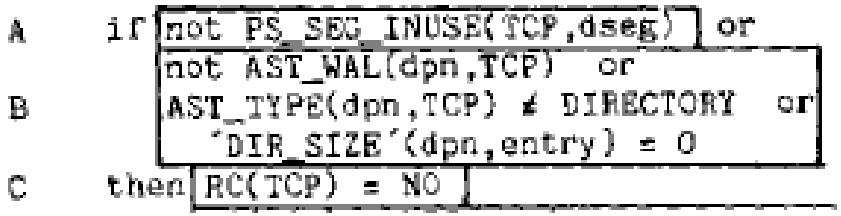
Statement	Read Levels	Write Levels
A	$PL(TCP)$	—
B	$PL(TCP), L(dpn)$	—
C	—	$PL(TCP)$
D	$PL(TCP), L(dpn)$	$PL(TCP), L(dpn)$
E (pn)	$PL(TCP)$	—
F (pn)	$L(apn)$	$L(pn)$
G $(pn, aproc)$	$PL(aproc)$	—
H $(pn, aproc)$	$PL(aproc), L(pn)$	$PL(aproc)$
J $(aproc)$	$PL(aproc)$	—
K $(aproc)$	$PL(aproc)$	$PL(aproc)$

Proof Case Example: Explanation of A, B, C



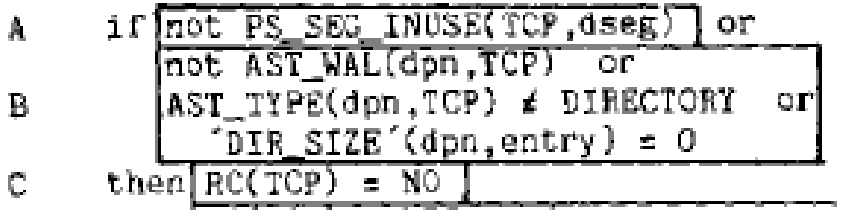
- Delete(dseg,entry)
  - Erases a segment from a directory
  - *Dseg* is directory segment, *entry* is index in directory
- (A) If the local segment number is not in use, or
- (B) If the process does not have write access to the directory or the directory entry is empty
- (C) Return “NO”

Proof Case Example: Function Explanation



- *dpn* is abbreviation for *PS_SEG(TCP,dseg)*
  - Directory path name
- **Active segment table (AST) is part of global state**
- **AST entry numbers must be invisible to avoid channel**
- PS_SEG(TCP,dseg) maps process-local segment numbers to active segment table AST entries
- PS_SEG_INUSE indicates whether or not an element in PS_SEG is in use
- AST_WAL is active segment table write access list

Proof Case Example: Proof Goal



Subject/ Case	Condi- tions	Ef- fects	Read Levels	Write Levels
S_1/I	A	C	$PL(TCP)$	$PL(TCP)$
S_1/II	$\bar{A}\ B$	C	$PL(TCP), L(dpn)$	$PL(TCP)$

- Second case: ¬A ∧ B ∧ C
  - PS_SEG_INUSE(TCP,dseg) = TRUE and
  - AST_WAL(dpn,TCP) = FALSE or… or …
- Prove second case does not violate *-property
- Process is reading from the directory and writing to the response code, so must prove: L(dir) ≤ L(RC)
- I.e., **L(dpn) ≤ PL(TCP)**

Proof Case Example: Proof

Two general relations suffice to complete the proof of the inequality:

R1: $(\forall proc)(\forall seg)$ if $PS\_SEG\_INUSE(proc, seg) = TRUE$
then $AST\_CPL(PS\_SEG(proc, seg), proc) = TRUE$.

R2: $(\forall pn)(\forall proc)$ if $AST\_CPL(pn, proc) = TRUE$
then $L(pn) \leq PL(proc)$.

- R1: If PS_SEG_INUSE is true then it must be the case that the process is in the AST "connected process list" (CPL) for that segment
- R2: If a process is in the AST_CPL then it must be the case that **`L(pn) ≤ PL(proc)`**
- Relations proven inductively over all operations

GEMSOS Verification

- PDP 11/45 verification before TCSEC
- GEMSOS developed to meet TCSEC class-A1
- Gemini Trusted Network Processer (GTNP) developed to be TNI M-component (multilevel)
  - Based on GEMSOS
- Evaluation on GTNP
- This paper, however, about GEMSOS TCB only

GEMSOS A1 Formal Verification Process

- FSPM, FTLS written in InaJo specification language
- BLP BST proven using FDM theorem prover
  - FSPM was not “pure” BLP, but the GEMSOS interpretation of BLP
- Conformance of FTLS to model also proven
- FTLS also used for code correspondence and covert storage channel analysis

Value of Formal Verification Process

- "Provided formulative and corrective guidance to the TCB design and implementation"
- I.e., just going through the process helped prevent and fix errors in the design and implementation
- Required designers/developers to use clean designs
  - So could be more easily represented in FTLS
  - Prevents designs difficult to evaluate and understand

GEMSOS TCB Subsets

- Ring 0: Mandatory security kernel
- Ring 1: DAC layer
- Policy enforced at TCB boundary is union of subset policies

Each Subset has its own FTLS and Model

- Each subset was verified through a separate Model and FTLS
- Separate proofs, too
- TCB specification must reflect union of subset policies

Where in SDLC?

- Model and FTLS written when interface spec written
- Preliminary model proofs, FTLS proofs, and covert channel analysis performed when implementation spec and code written
- Code correspondence, covert channel measurements, and final proofs performed when code is finished

- Formal verification went on simultaneously with development

Goal of GEMSOS TCB Verification

- To provide assurance that TCB implements the stated security policy
- Through chain of formal and informal evidence
  - Statements about TCB functionality
  - Each at different levels of abstraction
    - Policy
    - Model
    - Specification
    - Source
    - TCB itself (hardware and software)
  - Plus assertions that each statement is valid wrt next more abstract level

Chain of Verification Evidence

- Notes:
  - Model-to-policy argument is informal
  - Spec to model argument is both formal and informal
  - Source to spec argument is code correspondence
  - TCB to source means HW and compiler validation
    - I.e., object code
    - Considered "beyond state of the art"

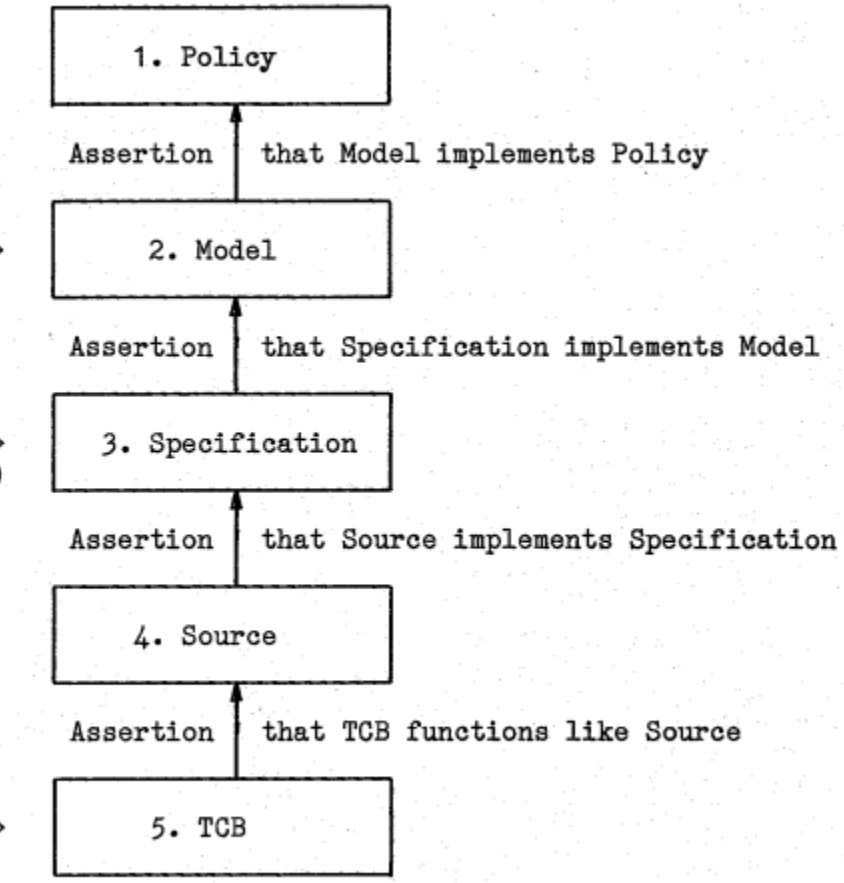


Figure 1. Chain of Verification Evidence

Model

- Mathematical statement of access control policy
- “Interpretation” of BLP
- Security defined as axioms
- Must prove all model transforms preserve axioms
  - SSC
  - *-property
  - (and probably others, as with PDP 11/45)
- Proof of model shows model upholds policy

Key Characteristic of Model

- Not just formal statement of policy or functions
- A model of a reference monitor
  - "Linchpin" of security argument
- If show that TCB satisfies reference monitor model then have shown that it is secure
  - Implies that anything outside TCB cannot violate policy
- What if did not model reference monitor?
  - May be "correct" wrt functions, but not necessarily secure

FDM “Levels”

- I.e., levels of abstraction
- InaJo language has method of formally mapping elements of one level to the elements of the next level
  - Top level: Model
  - Second level: FTLS

FTLSs

- One each for kernel and for TCB
- Exceptions, error messages, and effects visible at interface
- Transform for each call and
- Transforms for HW “read” and “write” operations
  - Other opcodes are irrelevant for access control security
- Proof maps each transform of FTLS to transform in model
- Each call specified as conditional statement
  - Last case contains any change statements
  - Exceptions specified in order of possible occurrence in code
    - Important for Covert Channel Analysis
  - Very end specifies everything else unchanged

Code Correspondence

- Three parts:
  1. Description of correspondence methodology
  2. Account of non-correlated source code
  3. Map between elements of FTLS and TCB code
- FTLS must accurately describe the TCB
- TCB must be valid interpretation of FTLS
- All security-relevant functions of TCB must be represented in FTLS
  - Prevent deliberate or accidental “trap door”

Example of Value of Formal Proof

- Subject is process/ring
- Subject can have range of access classes (trusted subject)
- Subjects in outer rings can have access class ranges "smaller" than subjects of the same process ir

```
Ring 3 Subject Range          READ WRITE
Ring 2 Subject Range      READ         WRITE
Ring 1 Subject Range  READ                 WRITE

(labels to the left dominate labels to the right)
(subject n is more privileged than subject n+1)
```

- Formal proof "stuck" trying to prove this

Example Formal Spec Detected Problem

- If range of subject in outer ring not within range of inner ring, move the outer ring access class to be within the range
- Original spec and code didn't take into account non-comparable access classes

```
dominates (ring_3_read_class, new_ring_2_read_class)
        then move (ring_3_read_class)
and
dominates (new_ring_2_write_class, ring_3_write_class)
        then move (ring_3_write_class)
```

- How to fix?

```
~dominates (new_ring_2_read_class, ring_3_read_class)
        then move (ring_3_read_class)
and
~dominates (ring_3_write_class, new_ring_2_write_class)
        then move (ring_3_write_class)
```

2nd Example Formal Spec Detected Problem

- Adjusting the access classes depends on the “move” function
- But it was found that the move function did not correctly ensure that the access class range of the outer ring subject was correct (i.e., that the “read” class dominated the “write” class)

Example of Value of Code Correspondence

- Code correspondence of kernel to spec found flaws in code:
  1. Access to segments in new child processes being checked using parent's privileges, not child's
  2. Segment descriptor in Local Descriptor Table not being set until segment brought into RAM
     - Not clear if this just meant inconsistent with model or was a real security problem

Example of Value of Covert Channel Analysis

- Two unexpected covert storage channels discovered
- Both related to “dismount_volume” call
- Dismount_volume used to (temporarily) remove set of segments from the segment structure
- Originally, any process whose access class range spanned range of volume could dismount the volume
- What if volume has only Unclassified segments?
  - TS process has made_known some of those segments
  - Unclassified process tries to dismount the volume, but gets an error message
- Fix?
  - Require caller’s range from volume low to sys-high

2nd Covert Channel

- Order of error checking
- Errors about volume could be reported to the calling subject even if subject did not have access to dismount the volume
- Fix: check label range before returning errors related to volume attributes

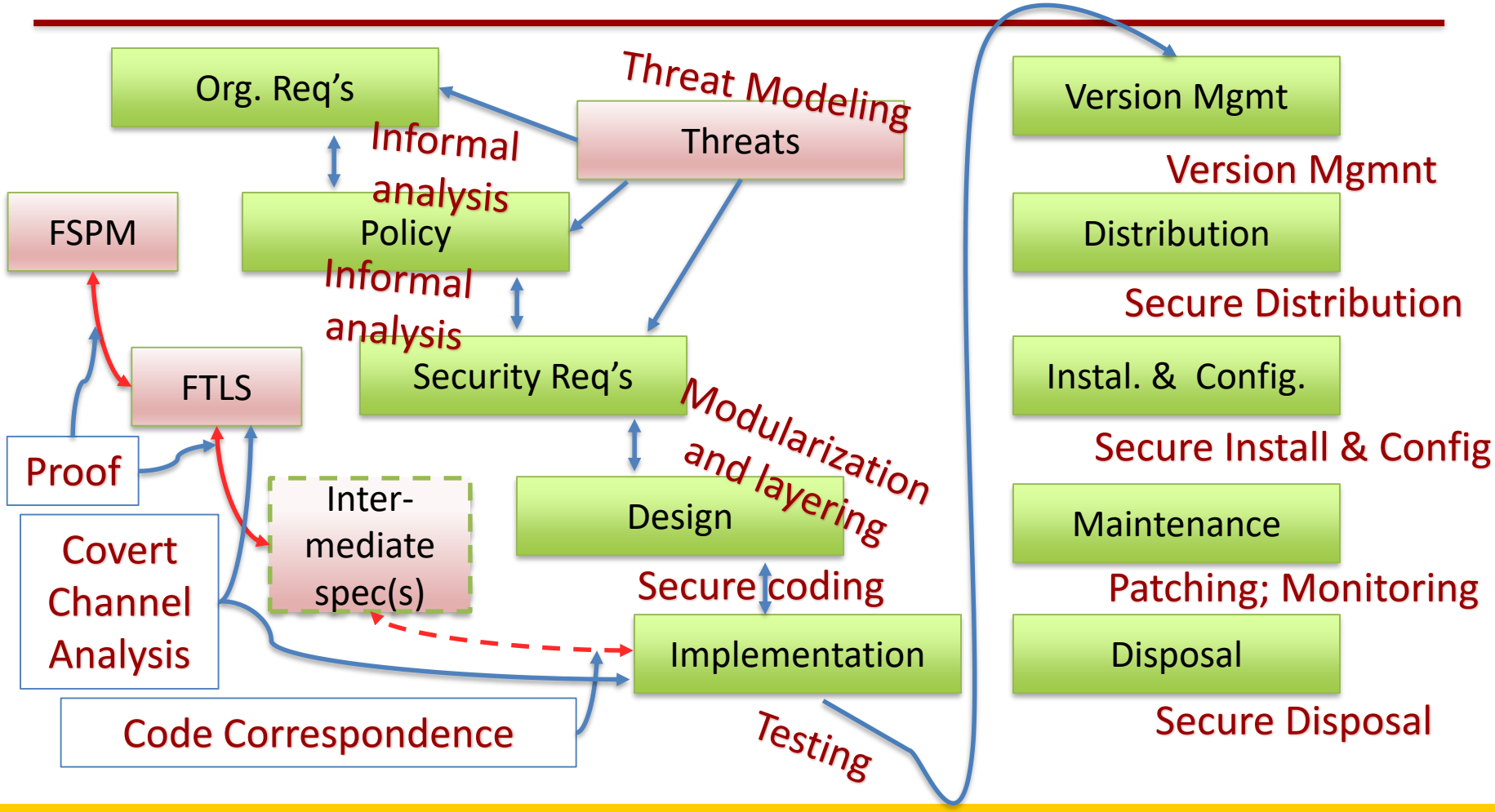
INF523: Assurance in Cyberspace Applied to Information Security

Covert Channel Analysis

Prof. Clifford Neuman

Lecture 10
27 October 2017

"Assurance Waterfall"



Reading for This Time

- Bishop book, Chapter 17 *Confinement Problem*
- *Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels*, Richard Kemmerer, 1983
- *Covert Flow Trees: A Visual Approach to Analyzing Covert Storage Channels*, Richard Kemmerer, 1991
- *An Entropy-Based Approach to Detecting Covert Timing Channels*, Steven Gianvecchio and Haining Wang, 2011

Covert Channels – TCSEC Definition

- A communication channel that allows a process to transfer information in a manner that violates the system's security policy.
  - Source: NCSC-TG-030 A Guide to Understanding Covert Channel Analysis of Trusted Systems (“light pink book”)

Covert Channels – Better Definition

- Given a nondiscretionary (mandatory) security policy model M and its interpretation I(M) in an operating system, any potential communication between two subjects I(Sh) and I(Si) of I(M) is covert if and only if any communication between the corresponding subjects Sh and Si of the model M is illegal in M.
  - Source: C.-R. Tsai, V.D. Gligor, and C.S. Chandersekaran, “A formal method for the identification of covert storage channel in source code”, 1990

Observations

- Covert channels are irrelevant for DAC policies because Trojan Horse can leak information via valid system calls and system can't tell what is illegitimate
  - Covert channel analysis only useful for trusted systems
- A system can *correctly implement* (interpret) a mandatory security policy model (like BLP) but *still not be secure* due to covert channels (violates *metapolicy*)
  - E.g., protects access to objects but not to shared resources
- Covert channels apply to integrity as much as secrecy
  - E.g., don't want low-integrity user to be able to influence high-integrity application through covert channel

Two Types of Covert Channels - TCSEC

- **Storage channel** "involves the direct or indirect writing of a storage location by one process [i.e., a subject of I(M)] and the direct or indirect reading of the storage location by another process."
- **Timing channel** involves a process that "signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process."
  - Source: TCSEC

Other Attributes Used in Covert Channels

- Timing: amount of time a computation took
- Implicit: control path a program takes
- Termination: does a computation terminate?
- Probability: distribution of system events
- Resource exhaustion: is some resource depleted?
- Power: how much energy is consumed?
- Any time SL can detect varying results that depend on actions by SH, that could form a covert channel

Storage Channel Example

- Attempted access by SL to a high level resource returns one of two error messages: Resource not found or Access denied. By modulating the status of the resource, SH can send a bit of information on each access attempt by SL.

- This is called a covert storage channel because SH is recording information within the system state.

Storage Channel Example, cont'd

- Consider a simple system that has READ and WRITE operations with the following semantics:
  - READ (S, O): if object O exists and LS ≥ LO, then return its current value; otherwise, return a zero
  - WRITE (S, O, V): if object exists O and LS ≤ LO, change its value to V; otherwise, do nothing
- These operations pretty clearly are acceptable instances of READ and WRITE for a BLP system

Source: Bill Young, Univ of Texas

Storage Channel Example, cont'd

- Add two new operations, CREATE and DESTROY to the system, with the following semantics:
  - CREATE (S, O): if no object with name O exists anywhere on the system, create a new object O at level LS ; otherwise, do nothing
  - DESTROY (S, O): if an object with name O exists and the LS ≤ LO, destroy it; otherwise, do nothing
- These operations seem to satisfy the BLP rules, but are they "secure"?

Storage Channel Example

In this system, a high level subject S_H can signal one bit of information to a low level subject S_L as follows:

S_H Transmits 0	S_H transmits 1
Create (S_H, F0)	do nothing
Create (S_L, F0)	Create (S_L, F0)
Write (S_L, F0, 1)	Write (S_L, F0, 1)
Read (S_L, F0)	Read (S_L, F0)
Destroy (S_L, F0)	Destroy (S_L, F0)

In the first case, S_L sees a value of 0; in the second case, S_L sees a value of 1. Thus, S_H can signal one bit of information to S_L by varying its behavior

Example Exploit

- To send 0:
  - High subject creates high object
  - Recipient tries to create same object but at low
    - Creation fails, but no indication given
  - Recipient gives different subject type permission to read, write object
    - Again fails, but no indication given
  - Subject writes 1 to object
    - Read returns 0

S_H Transmits 0	S_H transmits 1
Create (S_H, F0)	do nothing
Create (S_L, F0)	Create (S_L, F0)
Write (S_L, F0, 1)	Write (S_L, F0, 1)
Read (S_L, F0)	Read (S_L, F0)
Destroy (S_L, F0)	Destroy (S_L, F0)

Example Exploit

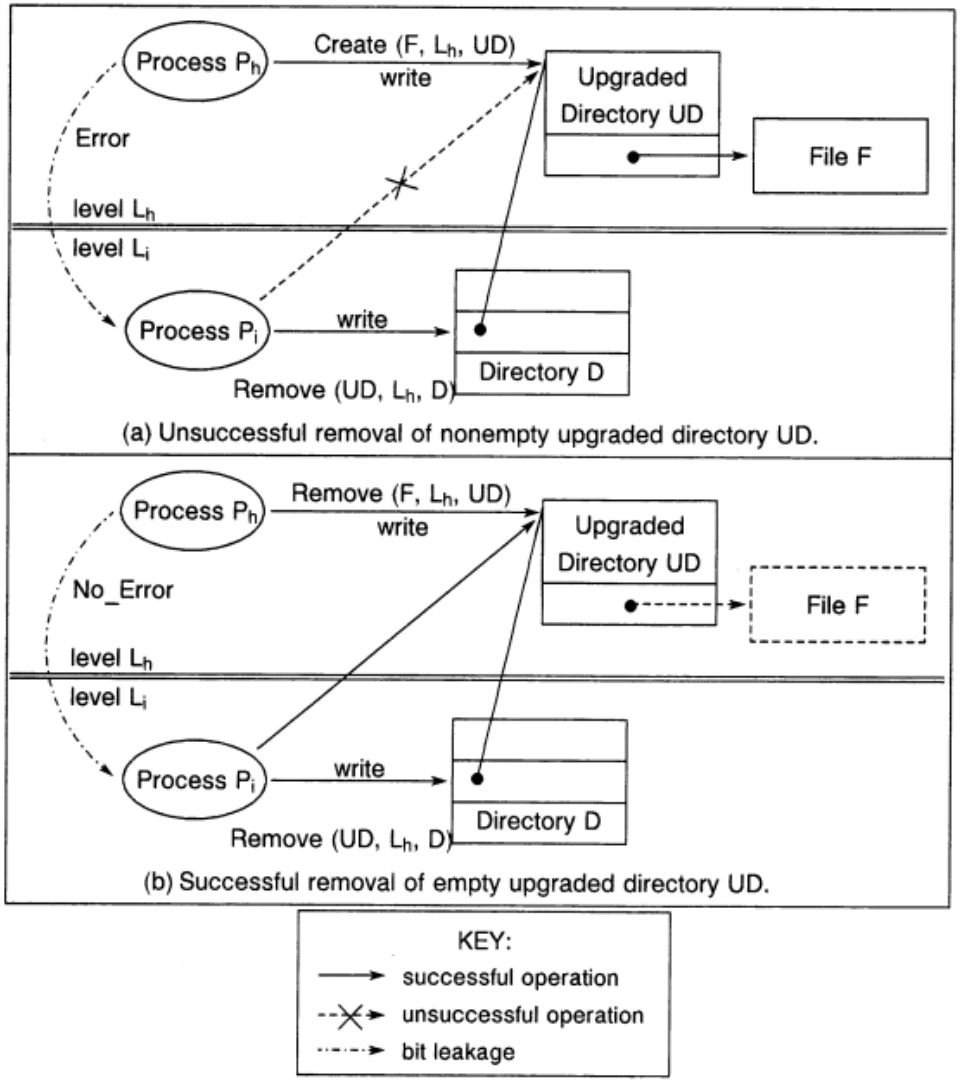
- To send 1:
  - High subject creates nothing
  - Recipient tries to create same object but at low
    - Creation succeeds as object does not exist
  - Recipient gives different subject type permission to read, write object
    - Again succeeds
  - Subject writes 1 to object
    - Read returns 1

S_H Transmits 0	S_H transmits 1
Create (S_H, F0)	do nothing
Create (S_L, F0)	Create (S_L, F0)
Write (S_L, F0, 1)	Write (S_L, F0, 1)
Read (S_L, F0)	Read (S_L, F0)
Destroy (S_L, F0)	Destroy (S_L, F0)

Another Example Storage Channel

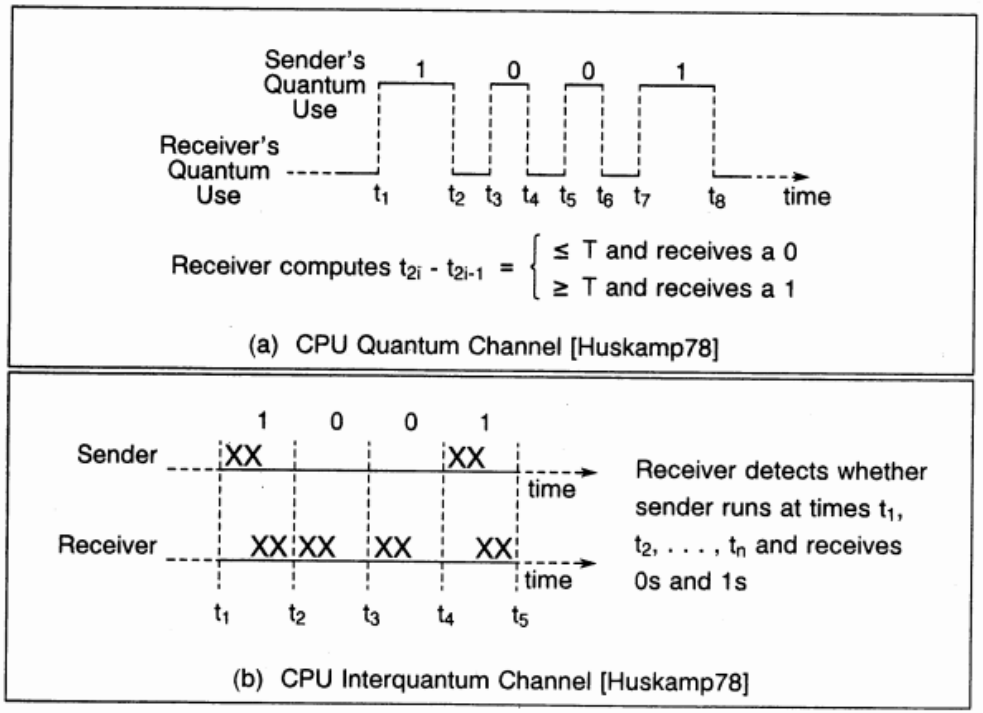
- Assume multi-level Unix
- Removal of non-empty directories in Unix is prohibited
- High-level subject can signal a low-level subject simply by manipulating the contents of the high-level directory
- What secure system have we studied that also had a storage object hierarchy? How did it avoid this problem?
- Multics permitted the removal of non-empty directories

Source: NCSC-TG-030 A Guide to Understanding Covert Channel Analysis of Trusted Systems ("light pink book")



Example Timing Channels

- CPU quanta is shared resource
- High signals low by amount of CPU time it uses
- First example:
  - Low counts time between its quanta
  - Long (>= T) equals 1
  - Short (< T) equals 0
- Second example:
  - High runs or not each quantum
  - High runs equals 1
  - High doesn't run equals 0



Another Example Timing Channel

- Developed by Paul Kocher
- This computes $x = a^z \bmod n$, where $z = z_0 \ldots z_{k-1}$

```
x := 1; atmp := a;
for i := 0 to k-1 do begin
  if z_i = 1 then
    x := (x * atmp) mod n;
  atmp := (atmp * atmp) mod n;
end
result := x;
```

- Length of run time related to number of 1 bits in z

Storage or Timing Channel?

- Processes H and L are not allowed to communicate, but they share access to a disk drive. The scanning algorithm services requests in the order of which cylinder is currently closest to the read head.
- Process H either accesses cylinder 140 or 160
- Process L requests accesses on cylinders 139 and 161
- Thus, L receives values from 139 and then 161, or from 161 and then 139, depending on H's most recent read
- Is this a timing or storage channel? Neither? Both?

Timing Channel

- Timing or storage?
  - Usual definition ⇒ storage (no timer, clock)
- Modify example to include timer
  - *L* uses this to determine how long requests take to complete
  - Time to seek to 139 < time to seek to 161 ⇒ 1; otherwise, 0
- Channel works same way
  - Suggests it's a timing channel; hence our definition
- Relative ordering channels are timing channels

Implicit Channel

- An implicit channel is one that uses the control flow of a program. For example, consider the following program fragment:

```
H := H mod 2;
L := 0;
if H = 1 then L := 1 else skip;
```

- The resulting value of L depends on the value of H.
- Language-based information flow tools can check for these kinds of dependencies in programming languages

Reading for Next Time

- D2L “Readings” folder:
  - NCSC-TG-030 A Guide to Understanding Covert Channel Analysis of Trusted Systems (“light pink book”), pp. 1-74

Side Channel vs. Covert Channel

- Covert channel
  - Intentional use of available channel
  - Intention to conceal its existence
- Side channel
  - Unintentional information leakage due to characteristics of the system operation
  - E.g., malicious VM gathering information about another VM on the same HW host
    - Share CPU, RAM, cache, etc.
    - This really can happen:

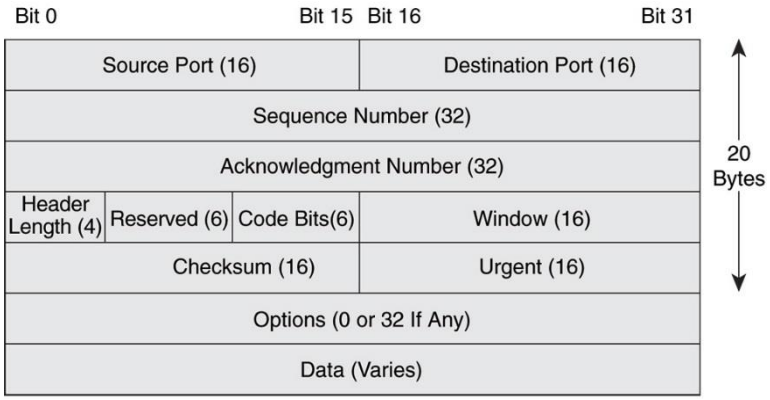
      Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In Proc. of the 2012 ACM Conference on Computer and Communications Cecurity (CCS '12). ACM, New York, NY, USA, 305-316. DOI=10.1145/2382196.2382230

Covert Channels in the Real World

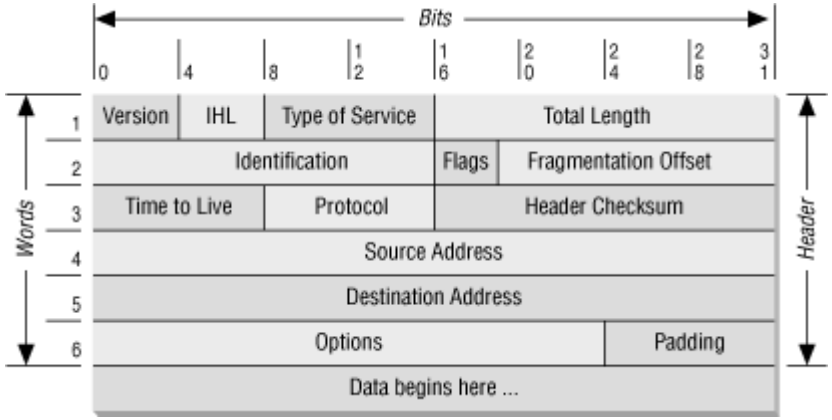
- Cloud IaaS covert channel
  - Side channel on the previous slide combined with encoding technique (anti-noise) and synchronization
  - Trojan horse on sending VM can signal another VM
- Trojan horse in your network stealthily leaking data
  - Hidden in fields of packet headers
  - Hidden in timing patterns of packets

Covert Storage Channel in Network Packets

- Many unused packet header (e.g., IP and TCP) fields
  - E.g., IP packet identification field
  - TCP initial sequence number field
  - TCP acknowledged sequence number field



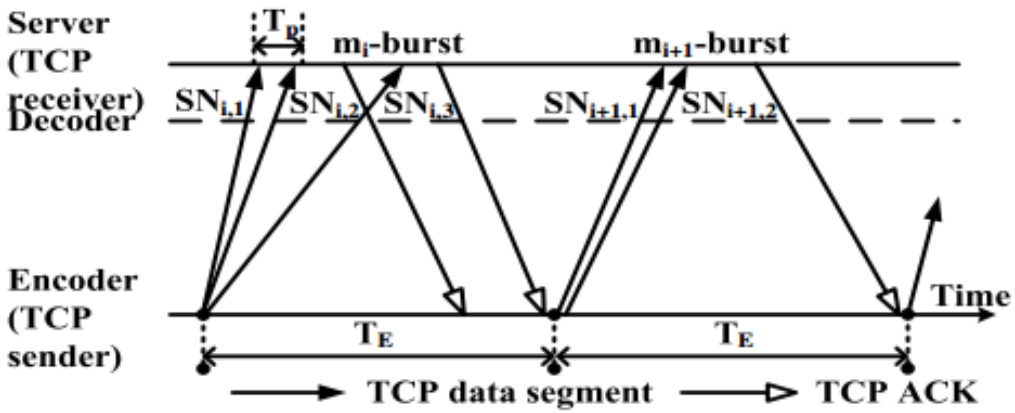
TCP Header



IP Header

Covert Timing Channel in Network Packets

- Can use timing interval (quantum)
- Can use varying inter-packet delays
- More sophisticated attacks look like normal traffic
  - E.g., size or number of packet "bursts"



Source: Xiapu Luo; Chan, E.W.W.; Chang, R.K.C., "TCP covert timing channels: Design and detection," Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on , vol., no., pp.420,429, 24-27 June 2008, doi: 10.1109/DSN.2008.4630112

Note the Implicit Mandatory Policy

- May enforce only DAC inside the system
- But *still have mandatory policy* with two clearances:
  - Inside, “us”
  - Outside, “them”
- Covert channel exfiltrates data from “us” to “them”
- So covert channels of interest for security even in systems that use DAC policy internally

Structure of a Covert Channel

- Sender and receiver must synchronize
- Each must signal the other that it has read or written the data
- In storage channels, 3 variables, abstractly:
  - Data variable used to carry data
  - Sender-receiver synchronization variable (ready)
  - Receiver-sender synchronization variable (finished)
    - Write-up is allowed, so may be legitimate data flow
- In timing channels, synchronization variables replaced by observations of a time reference

Example of Synchronization

- Processes *H*, *L* not allowed to communicate
  - But they share a file system
- Communications protocol:
  - *H* sends a bit by creating a file called *0* or *1*, then a second file called *send*
    - *H* waits until *send* is deleted before repeating to send another bit
  - *L* waits until file *send* exists, then looks for file *0* or *1*; whichever exists is the bit
    - *L* then deletes *0*, *1*, and *send* and waits until *send* is recreated before repeating to read another bit
- Creation and deletion of *send* are the synchronization variables

Example of Synchronization

- Recall the Create/Delete object calls channel

S_H Transmits 0	S_H transmits 1
Create (S_H, F0)	do nothing
Create (S_L, F0)	Create (S_L, F0)
Write (S_L, F0, 1)	Write (S_L, F0, 1)
Read (S_L, F0)	Read (S_L, F0)
Destroy (S_L, F0)	Destroy (S_L, F0)

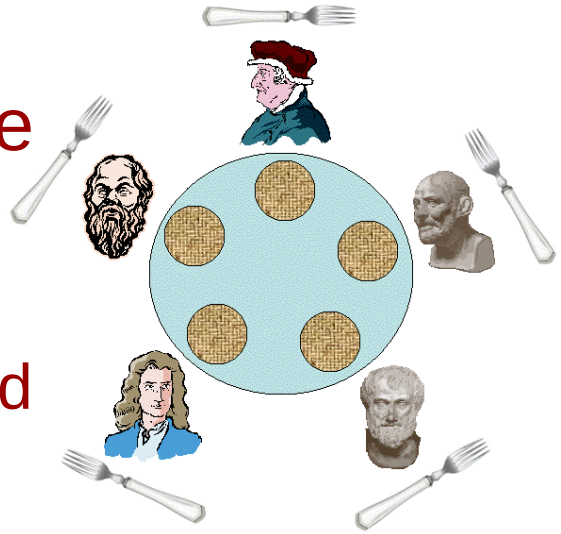
- How would you implement covert channel synchronization in this system.

Covert Channel Characteristics

- Existence: Is a channel present?
- Bandwidth: Amount of information that can be transmitted (bits per second)
- Noisiness: How much loss or distortion in the channel?

Noisy vs. Noiseless Channels

- Noiseless: covert channel uses resource available only to sender and receiver
- Noisy: covert channel uses resource available to others as well as to sender and receiver
  - E.g., other processes moving disk head in earlier example
  - Extraneous information is “noise”
  - Receiver must filter noise to be able to read sender’s “signal”

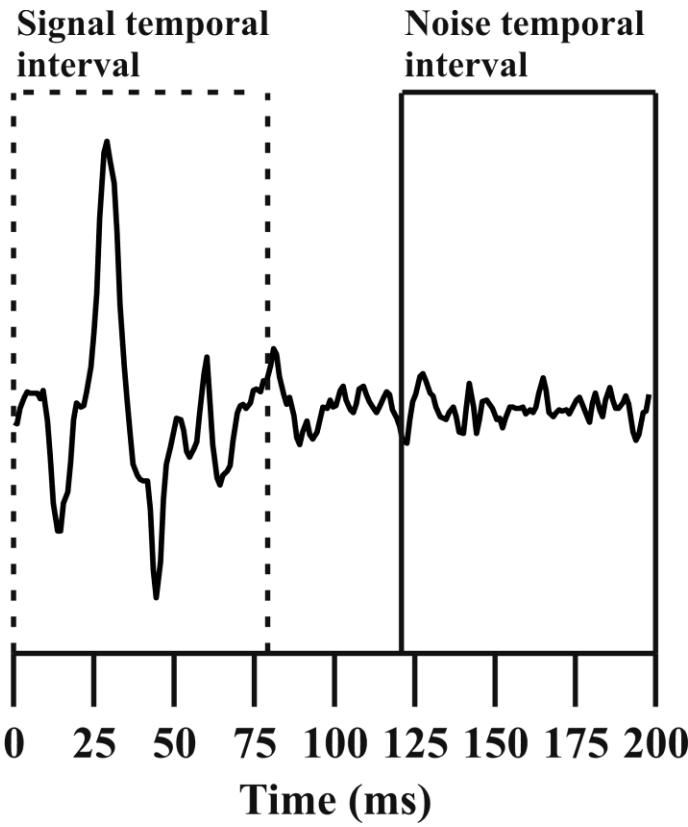


Objectives of Covert Channel Analysis

1. Detect all covert channels
   - Not generally possible
   - Find as many as possible
2. Eliminate them
   - By modifying the system implementation
   - Also may be impossible, or impractical
3. Reduce bandwidth of remaining channels
   - E.g., by introducing noise or slowing the time reference
4. Monitor any that still exceed the acceptable bandwidth threshold
   - Look for patterns that indicate channel is being used
   - I.e., intrusion detection

Noise and Filters

- If can't eliminate channel, try to reduce bandwidth by introducing noise
- But filters and encoding can be surprisingly effective
  - Need a lot of carefully designed noise to degrade channel bandwidth
  - Designers often get this wrong
- And added noise may significantly reduce system performance



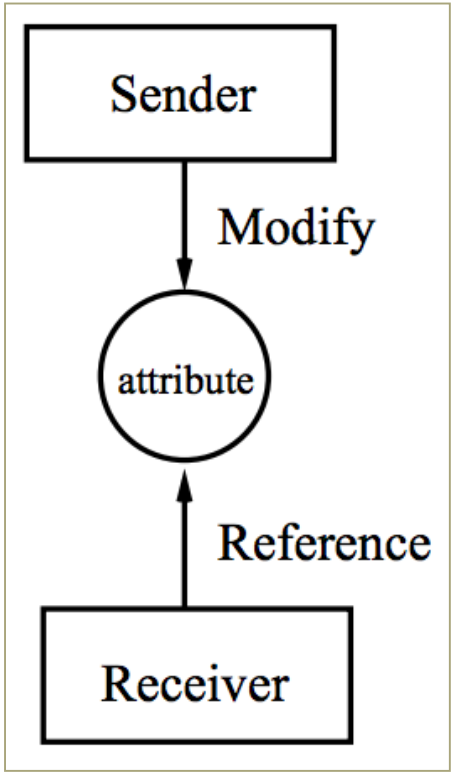
Step #1: Detection

- Manner in which resource is shared controls who can send, receive using that resource
  - Shared Resource Matrix Methodology
  - Covert flow trees
  - Non-interference

Covert Storage Channels, encore

Several conditions must hold for there to be a covert storage channel:

1. Both sender and receiver must have access to some attribute of a shared object
2. The sender must be able to modify the attribute
3. The receiver must be able to observe (reference) that attribute
4. Must have a mechanism for initiating both processes and sequencing their accesses to the shared resource



SRMM

- Technique developed by Richard Kemmerer at UCSB
- Build a table describing system commands and their potential effects on shared attributes of objects
  - An R means the operation "References" (provides information about) the attribute, under some circumstances.
  - An M means the operation "Modifies" the attribute, under some circumstances

Not useful for timing channels

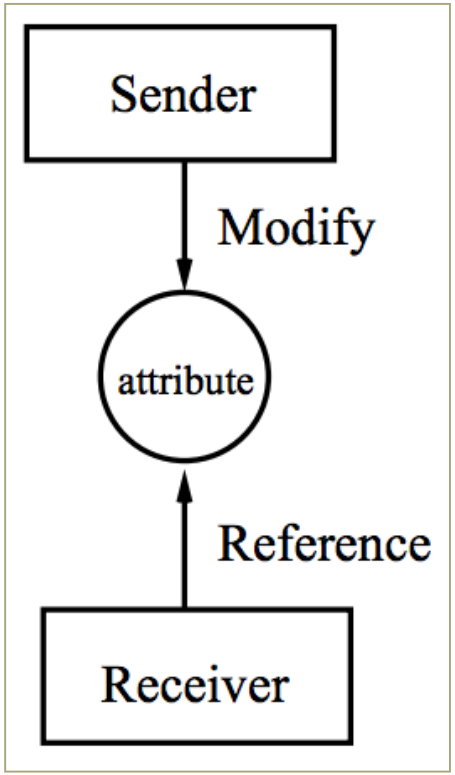
Attributes	READ	WRITE	DESTROY	CREATE
File existence	R		M	M
File size	R	M	M	M
File level	R		M	M

Using SRMM

If you see an R and M in the same row, that indicates a *potential* channel. Why potential?

SRMM doesn't identify covert channels, but suggests where to look for them

Any shared resource matrix is for a *specific system*. Other systems may have different semantics for the operations



SRMM Subtlety

- Suppose you have the following operation:
  - CREATE (S, O): if no object with name O exists anywhere on the system, create a new object O at level LS ; otherwise, do nothing
- For the attribute file existence, should you have an R or not for this operation or not? Consider this: after this operation, you know that the file exists. (Why?)
- That's not enough. It's not important that you know something about the attribute; what's important is that the operation tells you something about the attribute

SRRM Example: Unix

- Unix files have these attributes:
  - Existence, size, owner, group, access permissions (others?)
- Unix file operations to create, delete, open, read, write, chmod operations (others?)
- Homework: Fill in the shared resource matrix

	read	write	delete	create	open	chmod
existence						
size						
owner						
group						
Access permissions						

SRMM Example

- File attributes:
  - Existence, label
- File manipulation operations:
  - read, write, delete, create
    - Each returns completion code
  - create succeeds *if file does not exist*; gets creator's label
  - others require file exists, appropriate labels
- Subjects:
  - High, Low

	read	**write**	**delete**	**create**
existence	R	R	R, M	R, M
label	R	R	R	M

Example

- Consider existence row: has both R and M
- Let High be sender, Low receiver
- Create operation references and modifies existence attribute
  - Low can use this due to semantics of create
- Need to arrange for proper sequencing accesses to existence attribute of file (shared resource)

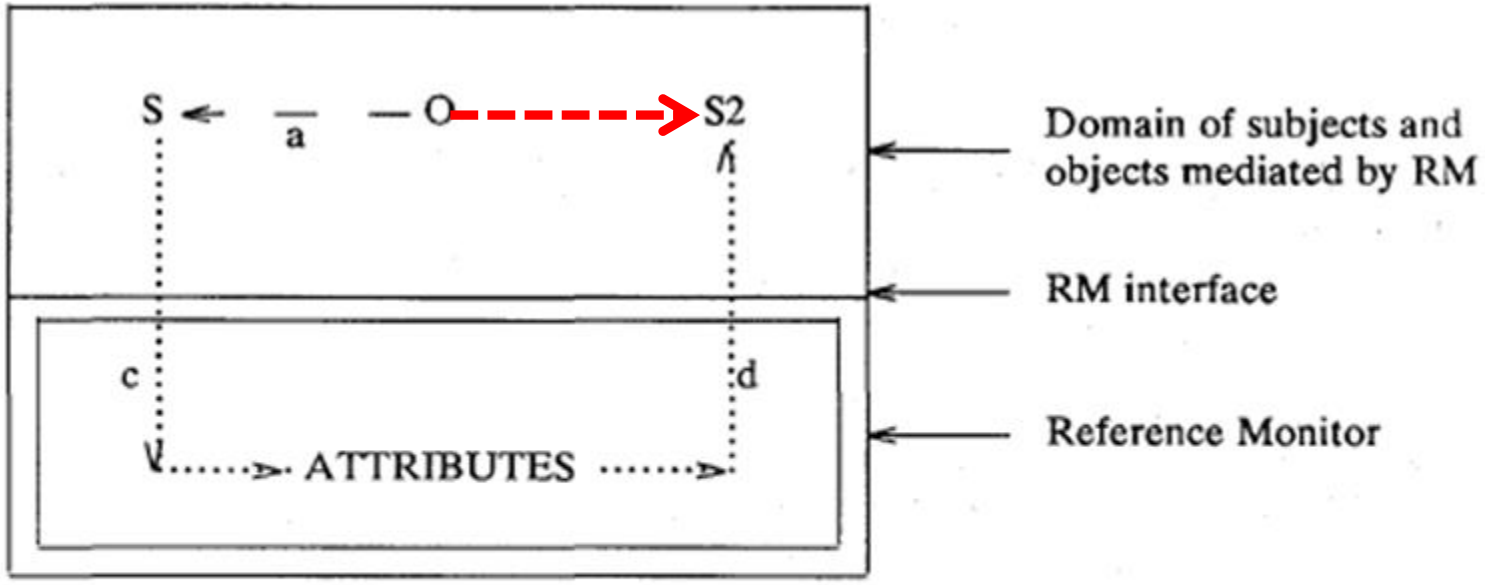
Use of Channel

- 3 files: *ready*, *done*, *1bit*
- Low creates *ready* at High level
- High checks that file exists
  - If so, to send 1, it creates *1bit*; to send 0, skip
  - Delete *ready*, create *done* at High level
- Low tries to create *done* at High level
  - On failure, High is done
  - Low tries to create *1bit* at level High
- Low deletes *done*, creates *ready* at High level

Transitive Closure

- Matrix initially shows direct flows
- Must also find indirect flows
- Transitively combine direct flows to find indirect flows and add to matrix
- A TCB primitive indirectly reads a variable y whenever a variable x, which the TCB primitive can read, can be modified by TCB functions based on a reading of the value of variable y
- When only informal specs of a TCB interface are available (not internal specs of each primitive), this step unnecessary since provides no additional information

Indirect Flows are Internal



S = subject — Level of S = syshi
O = object — Level of O = syshi
S2 = subject — Level of S2 = syslo
= Covert channel — Level of ATTRIBUTE = syshi
... = covert channel

Uses of SRM Methodology

- Applicable at many stages of software life cycle model
  - Flexbility is its strength
- Used to analyze Secure Ada Target
  - Participants manually constructed SRM from flow analysis of SAT model
  - Took transitive closure
  - Found 2 covert channels
    - One used assigned level attribute, another assigned type attribute

SRM Summary

- Methodology comprehensive but incomplete
  - How to identify shared resources?
  - What operations access them and how?
- Incompleteness a benefit
  - Allows use at different stages of software engineering life cycle
- Incompleteness a problem
  - Makes use of methodology sensitive to particular stage of software development

Non-interference Definition

- Intuitively:
  - Low-level user's "view" of the system should not be affected by anything that a high-level user does
- More formally:
  - Suppose L is a subject in the system
  - Now suppose you:
    1. run the system normally, interleaving the operations of all users
    2. run the system again after deleting all operations requested by subjects which should not be able to pass information to (interfere with) L
  - From L's point of view, there should be no visible difference
  - The system is "non-interference secure" if this is true of every subject in the system

Non-interference Implementation

- Non-interference is another policy, more abstract than BLP
- The enforcement mechanisms may be anything, including the BLP rules
- The more system state you add to the definition of “view”, can catch covert channels that uses that state

Limitations of Non-interference

- Non-interference is very difficult to achieve for realistic systems
- It requires identifying within the view function all potential channels of information
- Realistic systems have many such channels
- Modeling must be at very low level to capture many such channels
- Dealing with timing channels is possible, but difficult
- Very few systems are completely deterministic
- Some “interferences” are benign, e.g., encrypted files

TCSEC Bandwidth Guidelines

- Low bandwidths represent a lower risk
- Rate of one hundred (100) bps is considered "high“
  - not appropriate to call a computer system "secure"
- Rate < one (1) bps acceptable in most environments
- Audit any rate > one (1) bit in ten (10) seconds
- Trade-off system performance and CC bandwidth
  - Provide information for system developer to assess

Measuring Capacity

- Intuitively, difference between unmodulated, modulated channel
- E.g.,
  - Normal uncertainty in channel is 8 bits
  - Attacker modulates channel to send information, reducing uncertainty to 5 bits
  - Covert channel capacity is 3 bits
    - Modulation in effect fixes those bits

Mitigation of Covert Channels

- Problem: channels work by varying use of shared resources
- One solution:
  - Require processes to say what resources they need before running
  - Provide access to them in a way that no other process can access them
- Cumbersome!
  - Includes running (CPU covert channel)
  - Resources stay allocated for lifetime of process

Alternate Approach

- Obscure amount of resources being used
  - Receiver cannot distinguish between what the sender is using and what is added
- How? Two ways:
  - Devote uniform resources to each process
  - Inject randomness into allocation, use of resources

Uniformity or Randomness

- Uniformity: Subjects always use same amount of resources
  - Variation of isolation
  - Process can't tell if second process using resource
- Example: KVM/370 covert channel via CPU usage
  - Give each VM a time slice of fixed duration
  - Do not allow VM to surrender its CPU time
    - Can no longer send 0 or 1 by modulating CPU usage
- Randomness: Make noise dominate channel
  - Does not close it, but makes it useless

Randomness

- Example: MLS database
  - Probability of transaction being aborted by user other than sender, receiver approaches 1 -> very high noise
  - How to do this: have participants abort transactions randomly

Problem: Loss of Efficiency

- Fixed allocation constrains use and wastes resources
- Randomness wastes resources

- Policy question: Is the inefficiency preferable to the covert channel?

Example

- Goal: limit covert timing channels on VAX/VMM
- “Fuzzy time” reduces accuracy of system clocks by generating random clock ticks
  - Random interrupts take any desired distribution
  - System clock updates only after each timer interrupt
  - Kernel rounds time to nearest 0.1 sec before giving it to VM
    - Means it cannot be more accurate than timing of interrupts

Example

- I/O operations have random delays
- Kernel distinguishes 2 kinds of time:
  - *Event time* (when I/O event occurs)
  - *Notification time* (when VM told I/O event occurred)
    - Random delay between these prevents VM from figuring out when event actually occurred)
    - Delay can be randomly distributed as desired (in security kernel, it's 1–19ms)
  - Added enough noise to make covert timing channels hard to exploit

Improvement

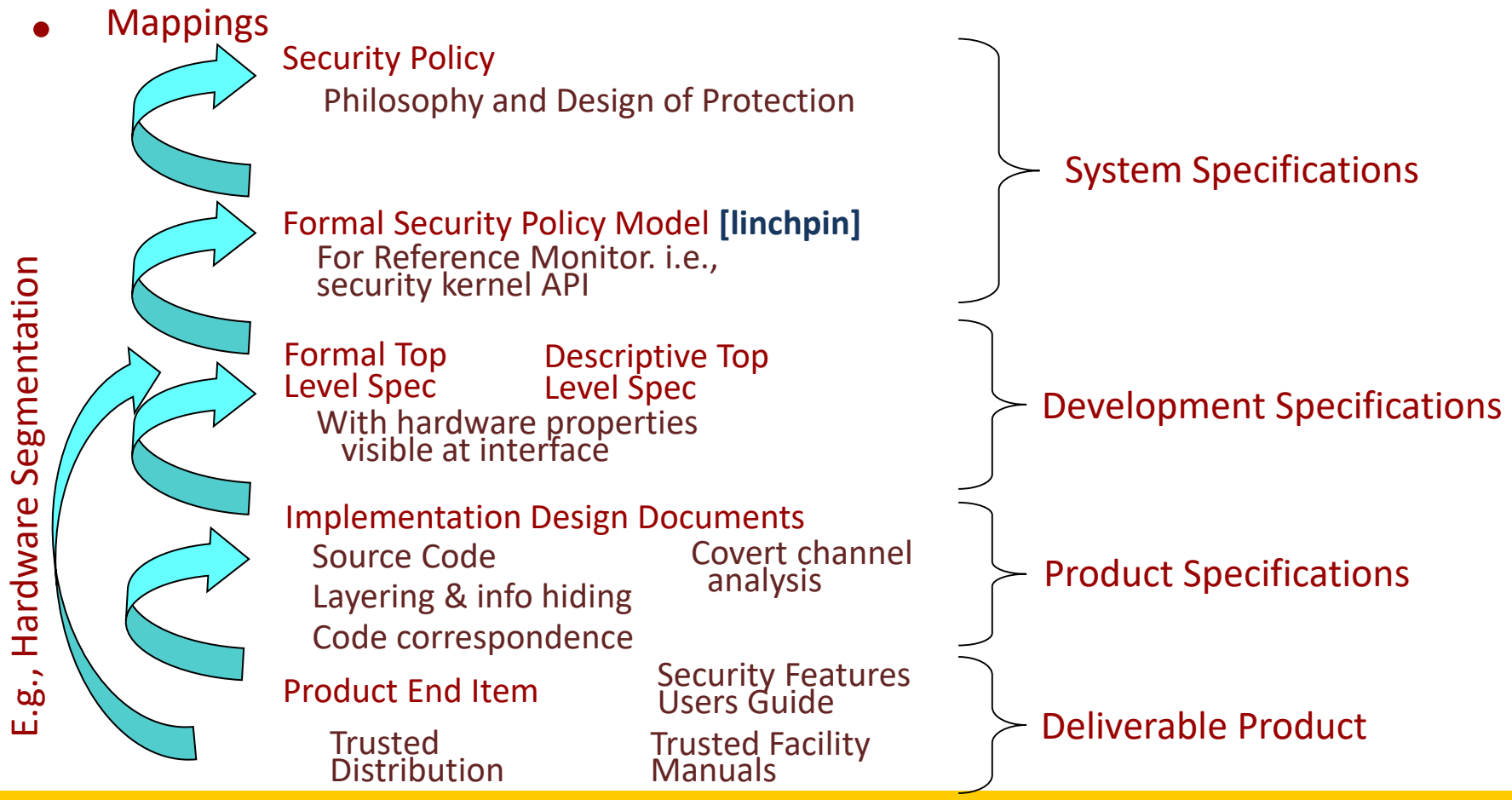
- Modify scheduler to run processes in increasing order of security level
  - Now we're worried about "reads up", so …
- Countermeasures needed only when transition from *dominating* VM to *dominated* VM
  - Add random intervals between quanta for these transitions

INF523:
Case Studies and Security Kernels

Professor Clifford Neuman

Lecture 10 CONT
26 October 2018

Systematic Kernel Engineering Process

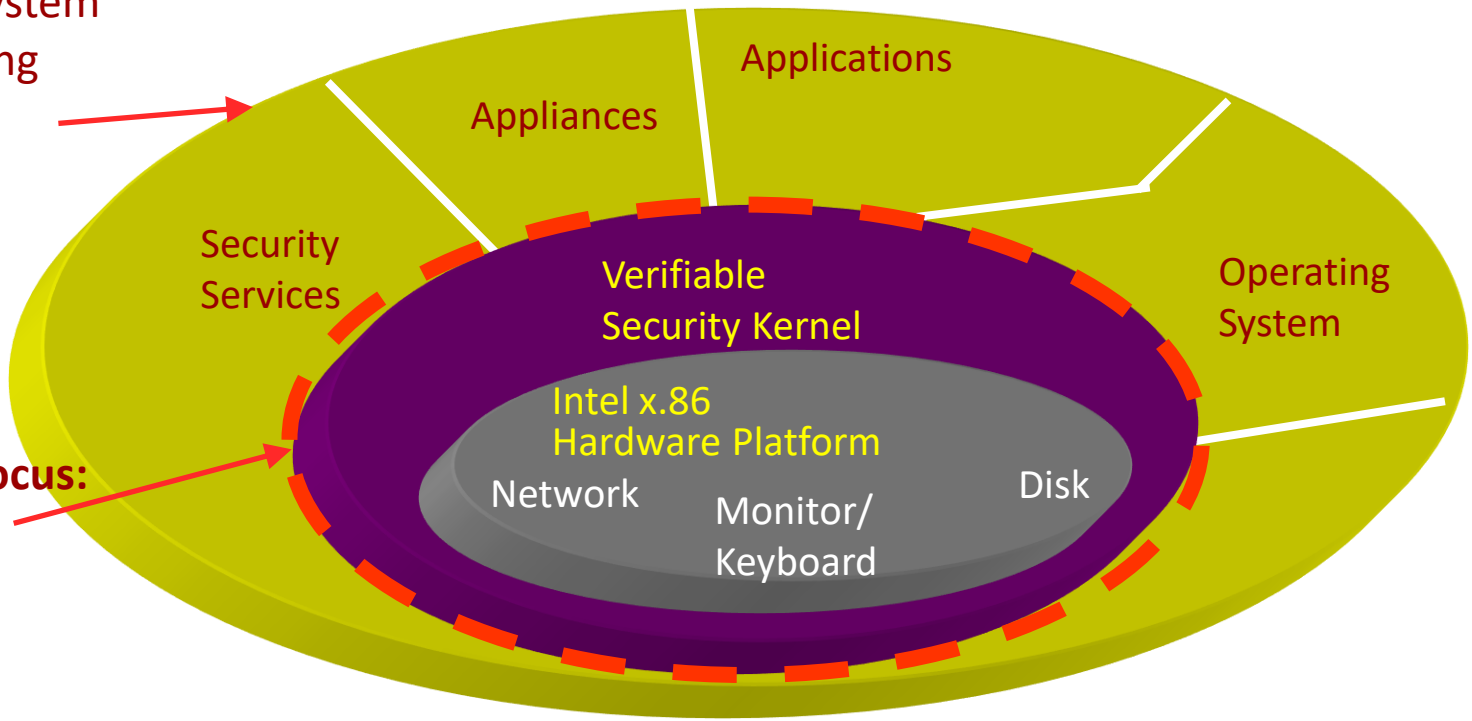


Only Proven Solution: Security Kernel

"The only way we know . . . to build highly secure software systems of any practical interest is the kernel approach."

-- ARPA Review Group, 1970s (Butler Lampson, Draper Prize recipient)

Secure System Engineering



INF 525 Focus: Verifiably Secure Platform

Truly a paradigm shift: no Class A1 security patches for kernel in years of use

Secure System Case Studies

CASE STUDIES

- GARNETS MLS File System Architecture
- NFS Artifice Demonstration Properties
- MLS Cloud NFS-Based Storage Design
- POSSIBLY:
  - Crypto Seal Guard Demonstration Concepts
  - Crypto Seals in RECON Guard for CIA

Kernel Implementation Strategies

- New operating system
  - Simple mapping of O/S features to SK features
  - Distinctive is lack of backward compatibility
- Compatible operating system (emulation)
- Emulate insecure operating system (ISOS)
  - Typically emulator runs in each process
    - Renders O/S calls into kernel calls
- Identical operating system (virtual machine)
  - Provides isolation, but not sharing, of memory
  - Kernel is virtual machine monitor (VMM)
    - Principal “objects” are virtual disks, not files
    - Subjects – kernel users and VMs

Designing a Security Kernel

- Most used highly secure technique
  - Not easy to build a security kernel
- SK is reference validation mechanism (RVM)
  - Defined as H/W and S/W that implements RM
- Most RMs implement multilevel security (MLS)
- Non-security-relevant functions managed by O/S
- Subject must invoke RM to reference **any** object
  - Basis for *completeness*
- Must have domain control, e.g., protection rings
  - Basis for *isolation* to block subversion
- SK software engineered for RM *verifiability*

Security Analysis of Trusted Systems

- Need independent 3rd party evaluation/analysis
- TCSEC/TNI security kernel evaluation factors
  - System architecture
  - Design specification & verification
  - Sensitivity label management
  - External interfaces
  - Human interfaces
  - Trusted system design properties
  - Security analysis
  - System use and management
  - Trusted system development and delivery

Secure System Design & Development

- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance
  - Operating systems services on kernel
- Asynchronous attacks and argument validation
- Protected subsystems
  - Static process, on-demand process, multiple domains
- Secure file systems
  - Alternate naming structures; unique identifiers

Secure System Design & Development

- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance

Designing a Security Kernel

- Most used highly secure technique
  - Not easy to build a security kernel
- SK is reference validation mechanism (RVM)
  - Defined as H/W and S/W that implements RM
- Most RMs implement multilevel security (MLS)
- Non-security-relevant functions managed by O/S
- Subject must invoke RM to reference **any** object
  - Basis for *completeness*
- Must have domain control, e.g., protection rings
  - Basis for *isolation* to block subversion
- SK software engineered for RM *verifiability*

GEMSOS Security Kernel Layering

- Segment make known is illustrative example
  - All modules in call chain are strictly layered
- Top gate layer is kernel API – implements FSPM
  - Receives call and completes entry to Ring 0
  - Parameters copied from outer ring to Ring 0
  - Entry point call to next layer
- Process-local modules at the top
  - Their “information hiding” data bases are per process
  - Code is shared with same PLSN by all processes
- Kernel-global modules
  - Kernel API “effects” reflect data all processes share

GEMSOS Make Known Example

(Applications, Kernel Gate Library)

Gate Layer	Process Local
Process Manager (PM)	
Upper Device Manager (UDM)	
Segment Manager (SM)	
Upper Traffic Controller (UTC)	Kernel Global
Memory Manager (MM)	
Inner Device Manager (IDM)	
Secondary Storage Manager	
Non-Discretionary Security Manager (NDSM)	
Kernel Device Layer (KD)	
Inner Traffic Controller (ITC)	
Core Manger (CM)	
Intersegment Linkage Layer (SG)	
System Library (SL)	

(Hardware)

Secure System Design & Development

- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance
  - Operating systems services on kernel

Operating System Layering Strategies

- New operating system: layer on security kernel
  - Simple mapping of O/S features to SK features
  - Distinctive is lack of backward compatibility
- Compatible operating system (emulation)
  - Emulate insecure operating system (ISOS)
  - Typically emulator runs in each process
    - Renders O/S calls into kernel calls
- Identical operating system (virtual machine)
  - Provides isolation, but not sharing, of memory
  - Kernel is virtual machine monitor (VMM)
    - Principal “objects” are virtual disks, not files
    - Subjects – kernel users and VMs

Secure System Design & Development

- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance
  - Operating systems services on kernel
- Asynchronous attacks and argument validation

Cross-Domain Asynchronous Attacks

- Hardware support for cross-domain validation
  - Pointer validation is particularly challenging
- Multiprocessor and multiprogramming issues
  - Multiple processes can access pointer argument
  - Time of check/time of use (TOC/TOU) problem
  - Safest to copy parameters to new domain before use
- OS must prevent changes to validation data
  - There are no generic solutions
  - May require appropriate locks inside OS
  - May require total atomic copy of data
    - Kernel support for this is valuable aid
- Examine I/O operations: are also asynchronous

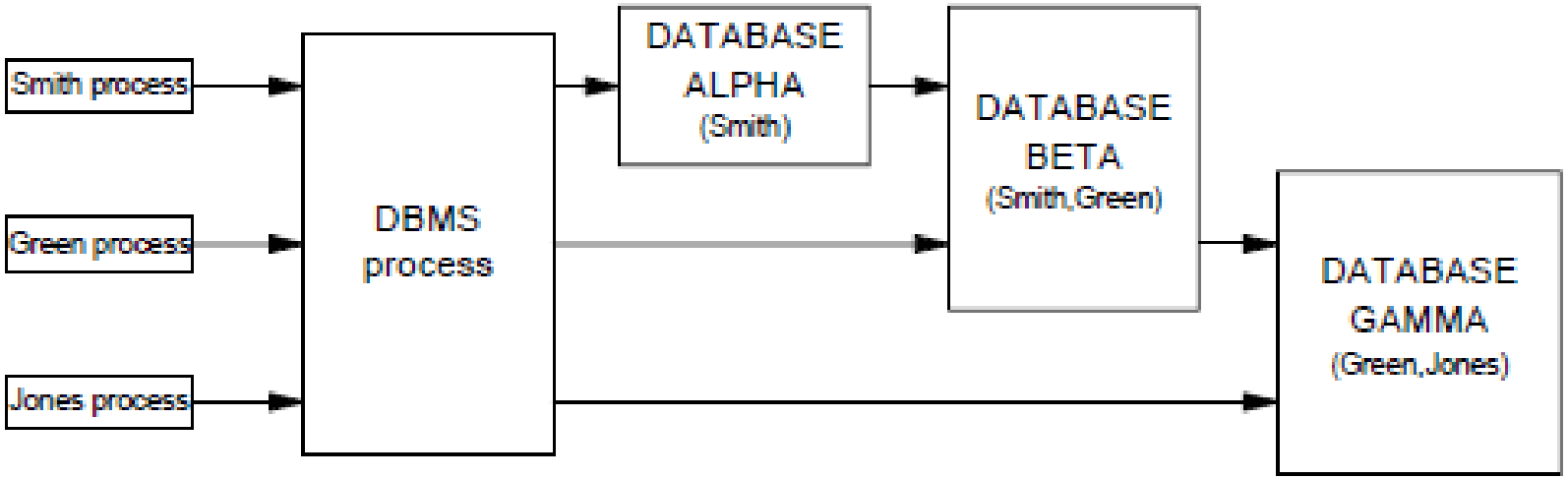
Synchronization for a Trusted System

- Useful operating system needs synchronization
  - Is usual and customary service applications expect
- Need synchronization across access classes
  - RVM must insure information flow meets MAC policy
- Mutexes and semaphores imply shared object
  - Read and write make not secure across access levels
- Use alternative NOT based on mutual exclusion
- Two kinds of objects are defined for computation
  - **Eventcount** to signal and observe progress
    - Primitives **advance(*E*), read(*E*), await(*E,v)***
  - **Sequencer** to assign an order to events occurring

Secure System Design & Development

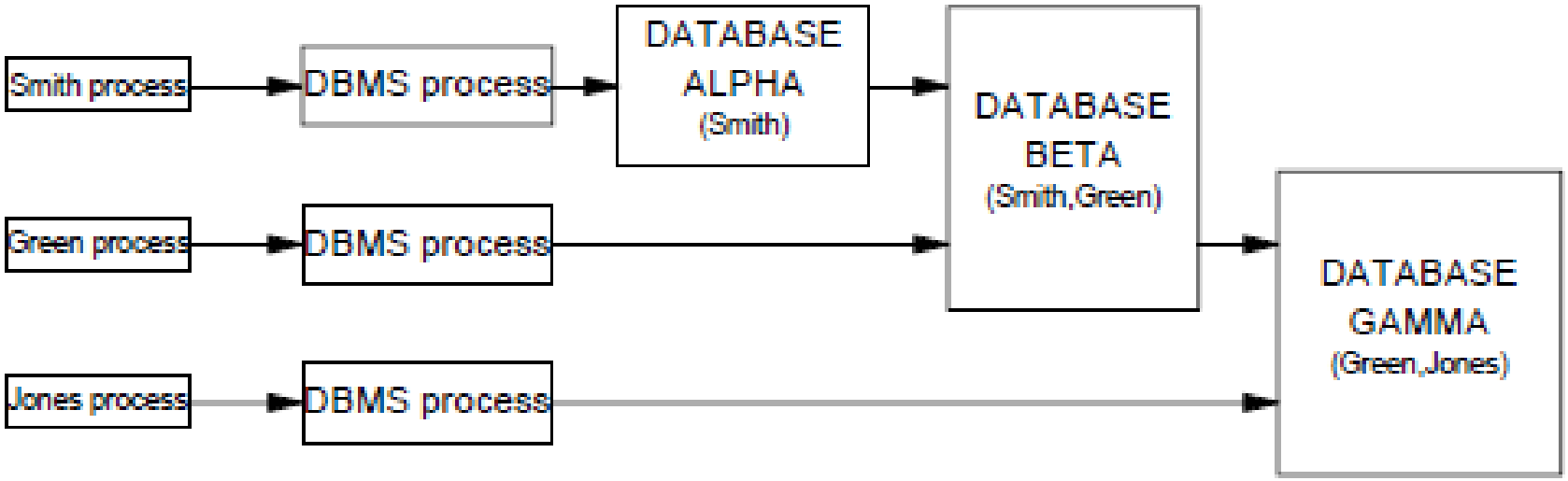
- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance
  - Operating systems services on kernel
- Asynchronous attacks and argument validation
- Protected subsystems
  - Static process, on-demand process, multiple domains

Protected Subsystem in Active Process



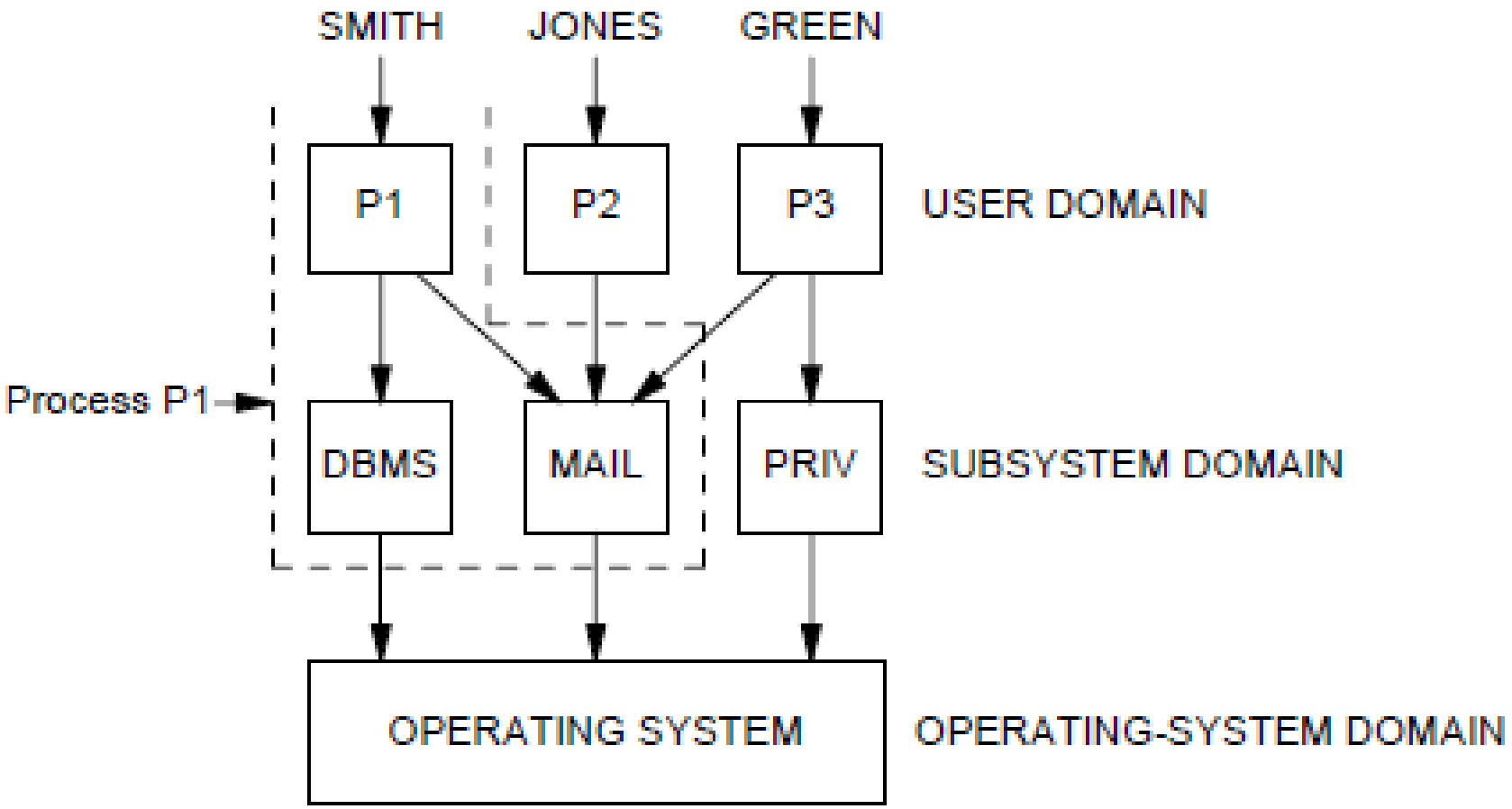
- DBMS runs as a process
  - Inherently no different from a user process
  - Normal OS access controls limit access to DBMS
- DBMS must control individual users' access
  - To different files and different portions of files

Protected Subsystem on Request



- Subsystem activated as a separate process
  - Each time it is needed
- While retaining its own identity
  - Separate from that of the invoking process

Mutually Suspicious Subsystems



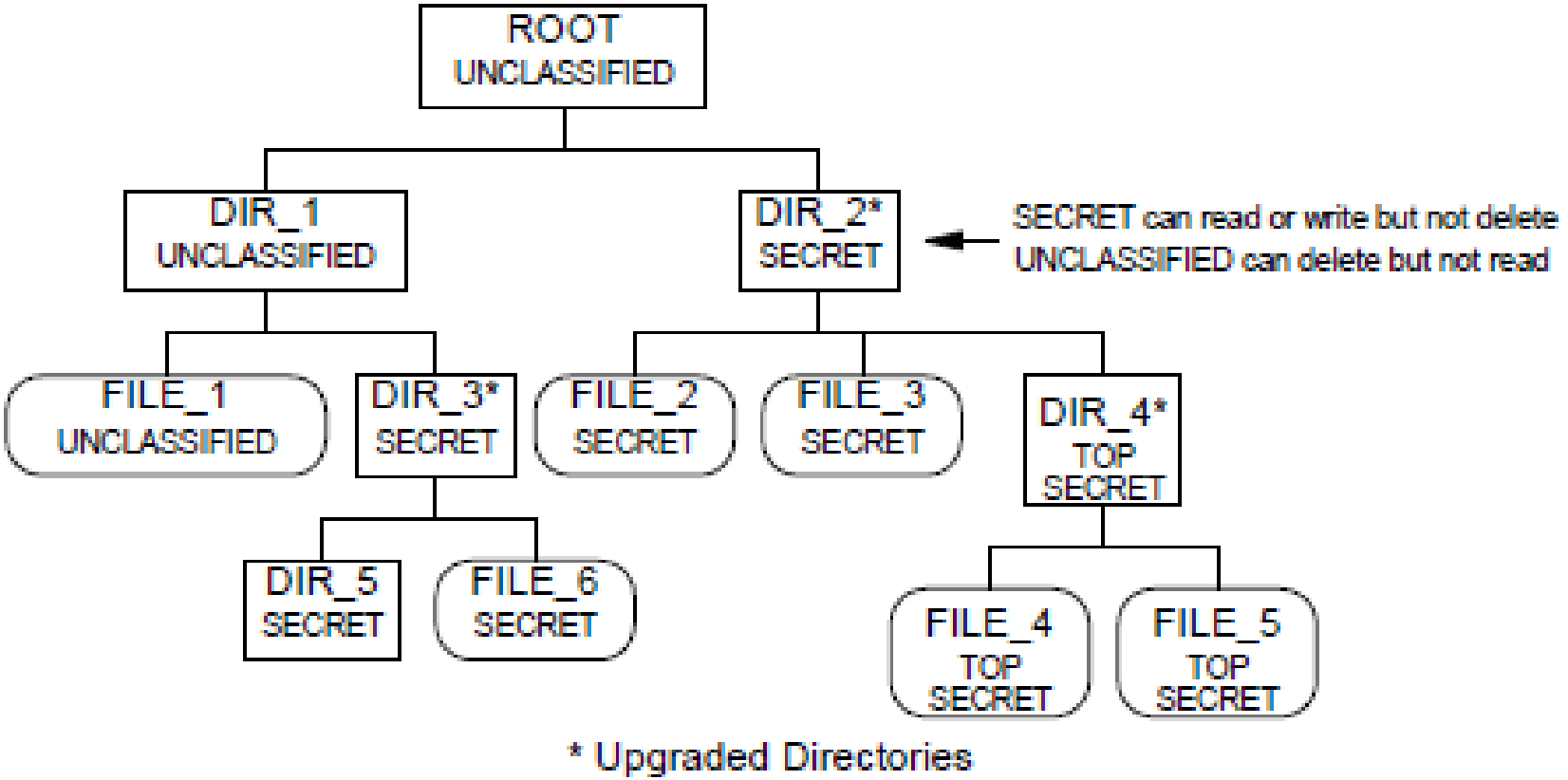
Management of SK Rings and Labels

- For a system, privileged services bound to ring
  - Static binding for given system architecture
  - Kernel is permanently bound to PL0
- Ring bracket (RB) associates object with domain
  - RB encoded in 3 ring numbers (RB1,RB2, RB3)
- General trusted system has at least 3 domains
  - Kernel, operating system, applications
- Non-discretionary is mandatory policy, i.e., MAC
- Each can be represented by access class label
  - Labels can be compared by dominance relation
  - Combine confidentiality, integrity, dominance domain

Secure System Design & Development

- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance
  - Operating systems services on kernel
- Asynchronous attacks and argument validation
- Protected subsystems
  - Static process, on-demand process, multiple domains
- Secure file systems
  - Alternate naming structures; unique identifiers

MLS Hierarchical File System



Security Kernel Objects

- Minimization of kernel
  - "Economy of mechanism"
  - "Significantly more complicated OS outside kernel"
    - Implies kernel cannot be compatible with insecure O/S
- All subjects need system-wide name for objects
  - Each subject must be able to identify shared object
  - "Flat" naming is classic covert channel example
- Object hierarchy naming
  - BLP hierarchy with "compatibility" meets need
  - Biba "inverse compatibility" for integrity needed
- Least common mechanism drives reuse
  - OS creates "directories" out of objects from kernel

Security Kernel Support for Segments

- Segmented instruction execution
  - Enforced for code is executing outside the kernel
- All memory addresses a pair of integers [s, i]
  - "s" is called the segment number;
  - "i" the index within the segment.
- Segment number is process local name (PLSN)
- Systems have similar kernel API to add segment
  - Kernel is invoked to “make known” a new segment
- Descriptor table defines process address space
  - Is a list of all the segments CPU can address
  - Must include code segment and stack segment

Secure System Architecture Summary

- Architectural considerations (Gasser chapter 11)
  - Applies to development of security kernels
  - As well as to their applications
- Operating-system layering
  - Promote structured design for kernel assurance
  - Operating systems services on kernel
- Asynchronous attacks and argument validation
- Protected subsystems
  - Static process, on-demand process, multiple domains
- Secure file systems
  - Alternate naming structures; unique identifiers

INF523

Introduction to MLS File System: GARNETS Case Study

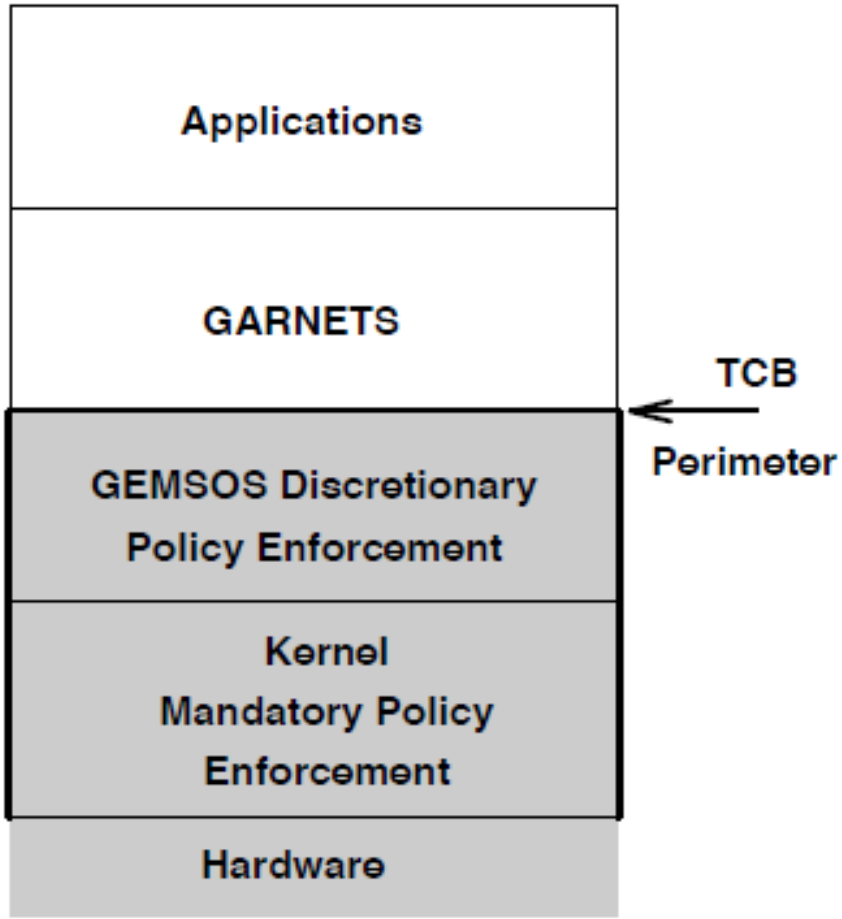
Professor Clifford Neuman

Lecture 12

GARNETS Example on Security Kernel

- Case study of broad application on top of TCB
- Security kernel TCB minimization severely limits
  - Can present only a primitive interface
- Lacks typical OS rich variety of functions
  - Argument that high assurance is “unusable”
- MAC enforcement constrains untrusted subjects
  - Argue renders application development “impossible”
- Analysis of GARNETS operating system
  - Gemini Application Resource and Network Support
  - Uses only TCB mechanism to provide interface
  - Interface is “friendly” and flexible

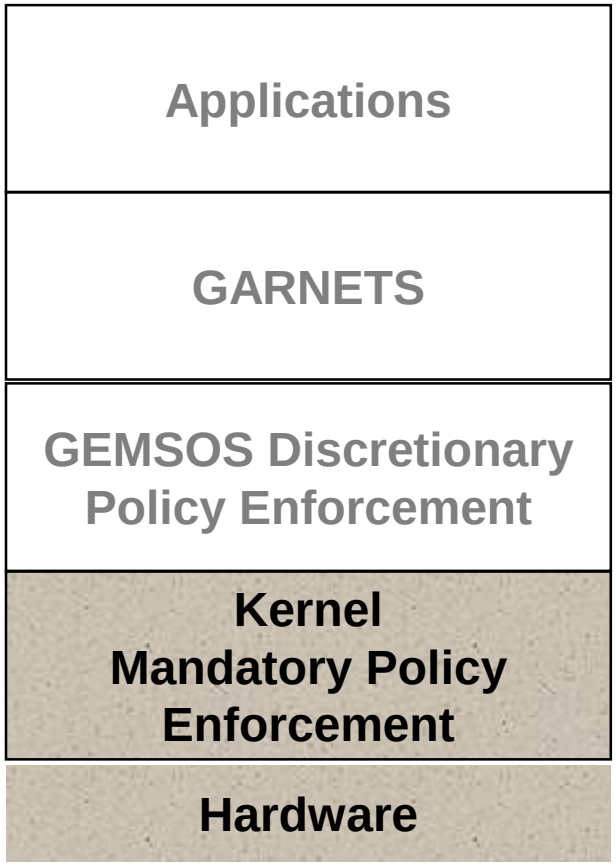
Standard GARNETS Architecture



Design Objectives

1. General purpose file system interfaces
   - Permit application libraries to be ported to interface
2. Both MAC and DAC exclusively from TCB
   - DAC subject on top of kernel provides “strong DAC”
3. File system is multilevel
   - Managed by single-level subjects
4. All file system operations are atomic
5. No read locks are used
   - Applications with can “read down” subject to DAC
6. Application access only GARNETS file system
7. GARNETS itself designed to meet Class B2

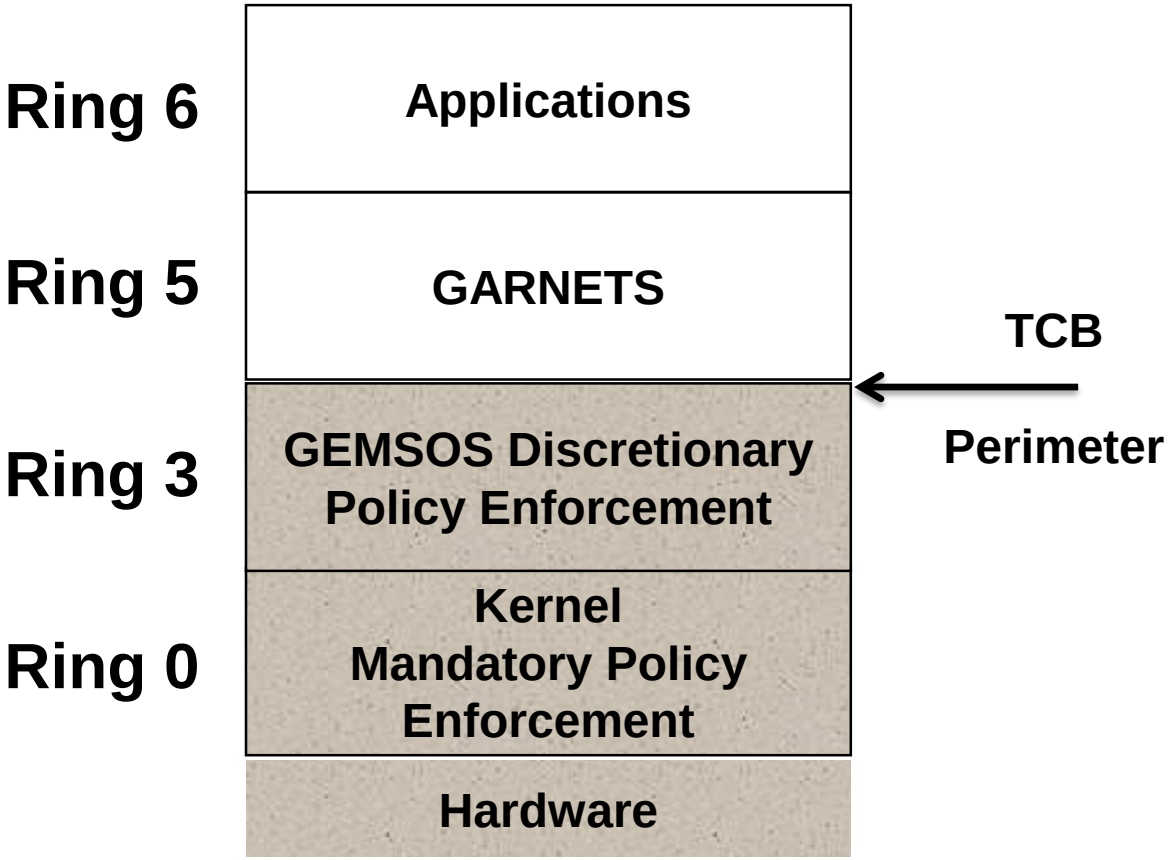
Overview of GARNETS Architecture



Kernel Exports Segments

- Segmented instruction execution
  - Enforced for code executing outside the kernel
- CPU memory addresses are pair of integers [s, i]
  - "s" is called the segment number;
  - "i" the index within the segment.
- Segment number is process local name (PLSN)
- TCB has API similar to kernel to add segment
  - Kernel is invoked to “make known” a new segment
- Descriptor table defines process address space
  - Is a list of all the segments CPU can address
  - DAC subject needs code segment and stack segment

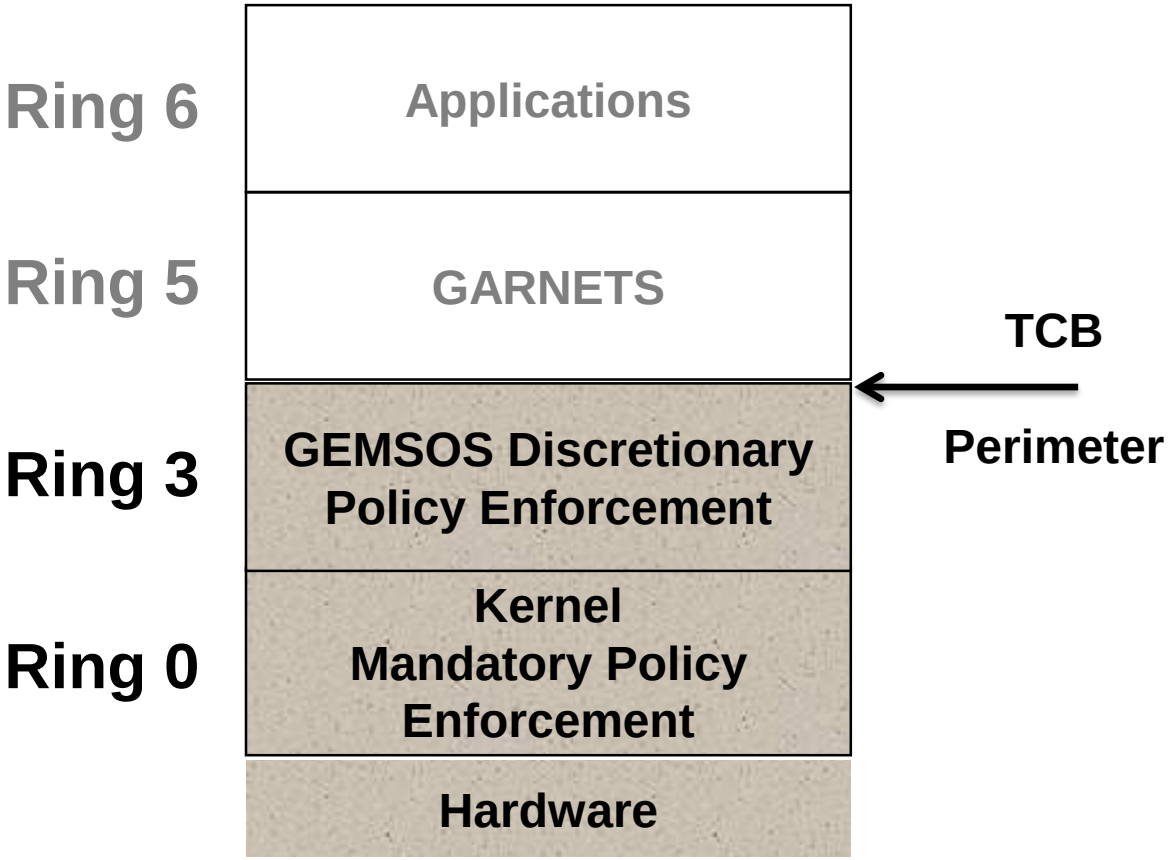
Domains in GARNETS Architecture



Kernel Creates (Domains) Rings

- For a system, privileged services bound to ring
  - Static binding for given system architecture
  - Kernel is permanently bound to PL0
- Ring bracket (RB) associates object with domain
  - RB encoded in 3 ring numbers (RB1,RB2, RB3)
- General trusted system has at least 3 domains
  - Kernel, operating system, applications
- Non-discretionary is mandatory policy, i.e., MAC
- Each subject represented by access class labels
  - Labels can be compared by dominance relation
  - Combine confidentiality, integrity, dominance domain

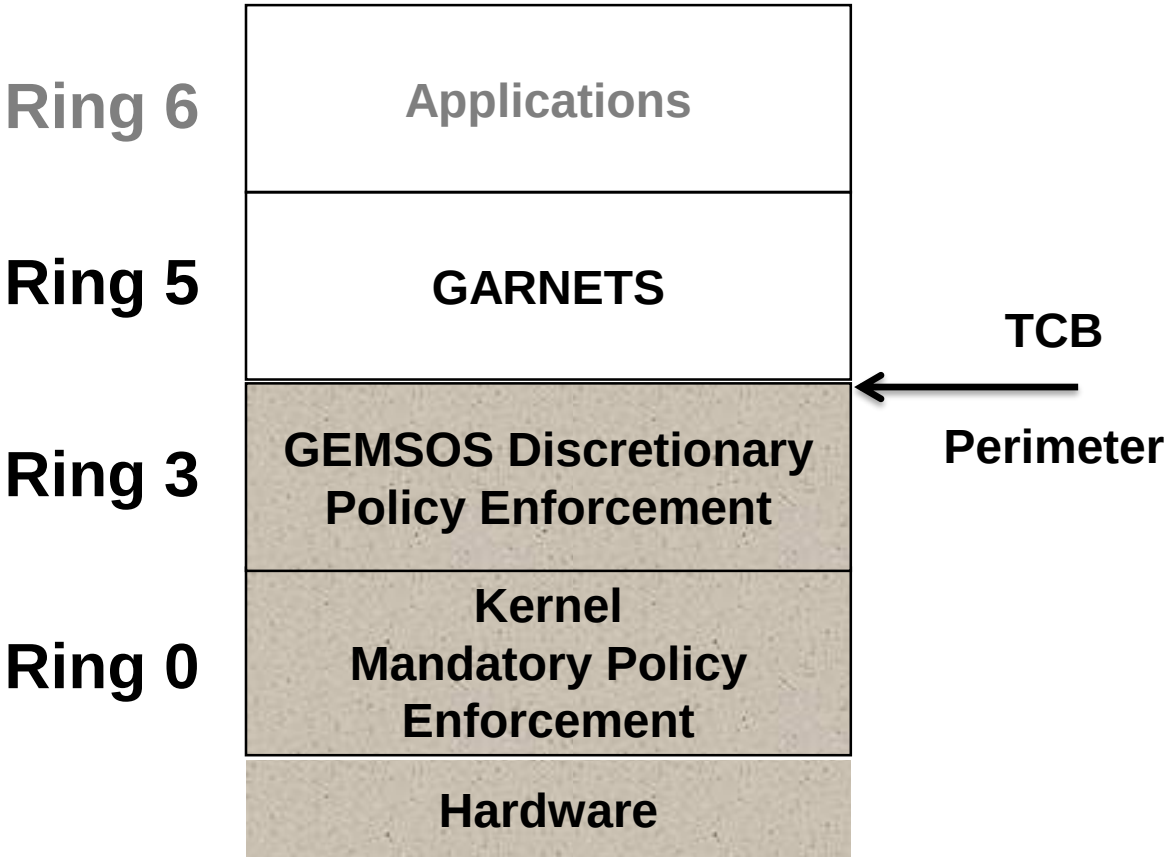
DAC TCB in GARNETS Architecture



DAC TCB Exports DACLS & Msegs

- Segments
  - Fundamental storage object
  - Loci of mandatory control
  - Processes simultaneously and independently share
- DAC Access Control Lists (DACLs)
  - A segment containing limited number of ACLs
  - Interpretively accessed object exported by TCB
  - Building block for GARNETS directories
- Multisegments (“msegs”) exported by DAC TCB
  - Collection of zero or more sements
  - Segments are accessibly only as elements of msegs
  - All its segments hierarchically related to single base

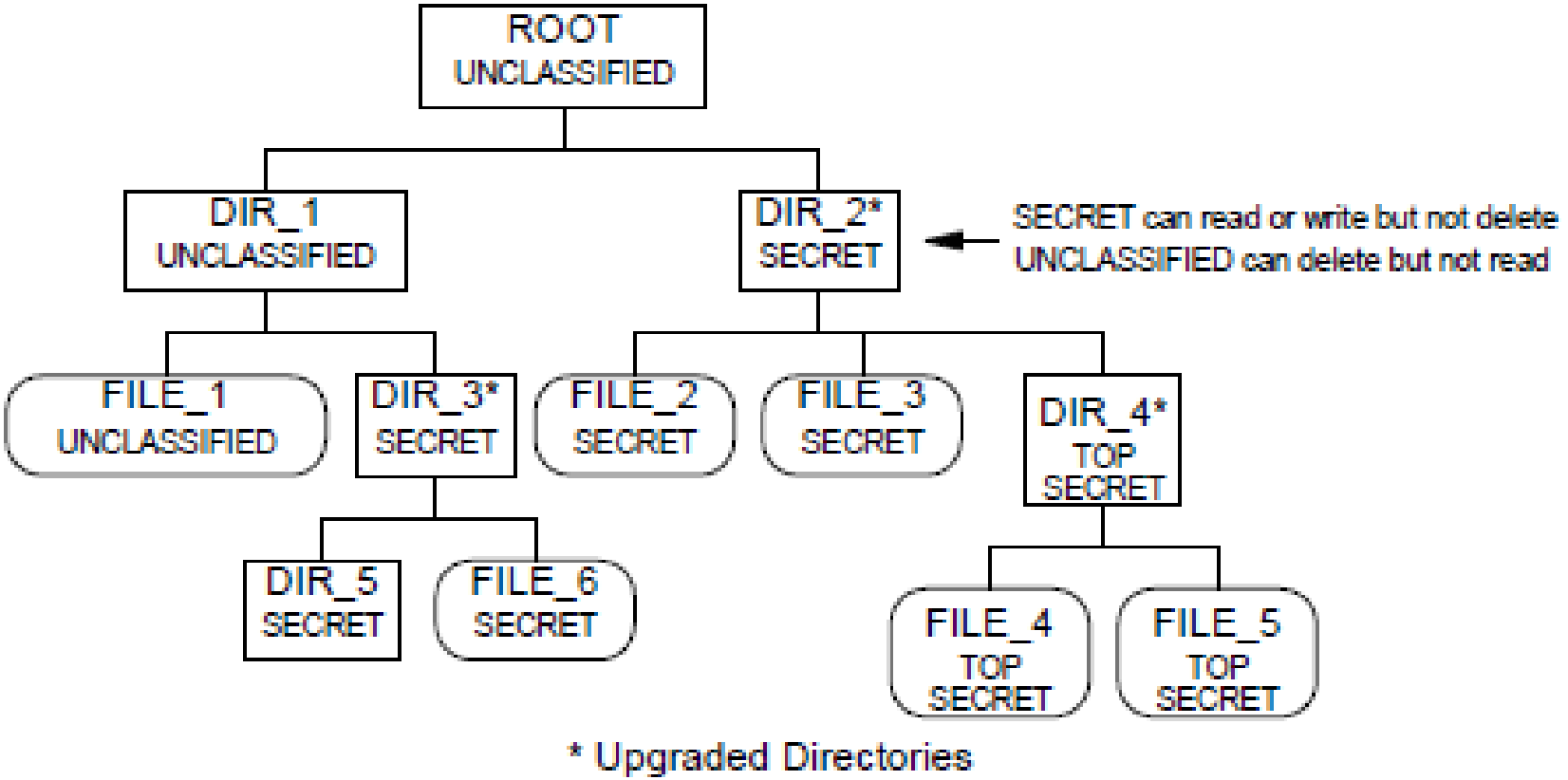
GARNETS in the Architecture



GARNETS Creates Files & Directories

- Directories
  - Rely only on TCB (not GARNETS) for access control
  - DAC access to an object based only on its ACL
- Named multisegments, distinct from files
  - Are namable directory entries
  - Its segments directly accessed via hardware
- Files interpretively access by applications
- File system built from three parallel trees
  - Directory tree of DACLs with mseg for dynamic data
  - File tree of DACLs which host file mseg for each file
  - Huge mseg with segments that mirror directory tree

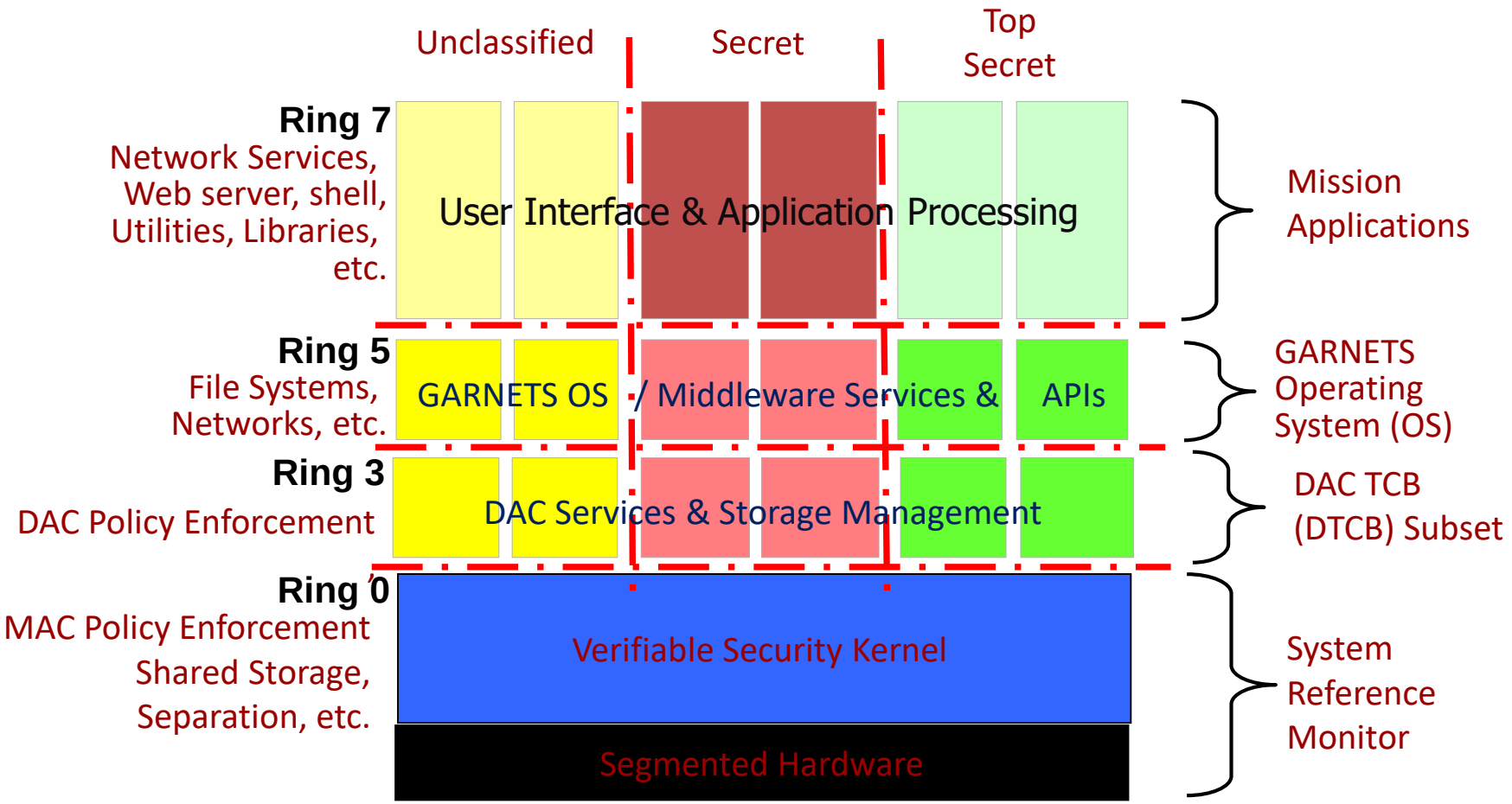
Gasser: MLS Hierarchical File System



Directory Properties

- Single-level directories
  - Contains information all at one access class
  - Subdirectory creation information is in parent
    - Names and creation dates
    - Visible to parent-level subjects
- Upgraded directories
  - Kernel forces compatibility property to be met
  - Dynamic information in upgraded directory itself
    - Time of last modification and directory contents
    - Visible only at the upgraded access class

TCB Subsets for GARNETS



Components of GARNETS Directory

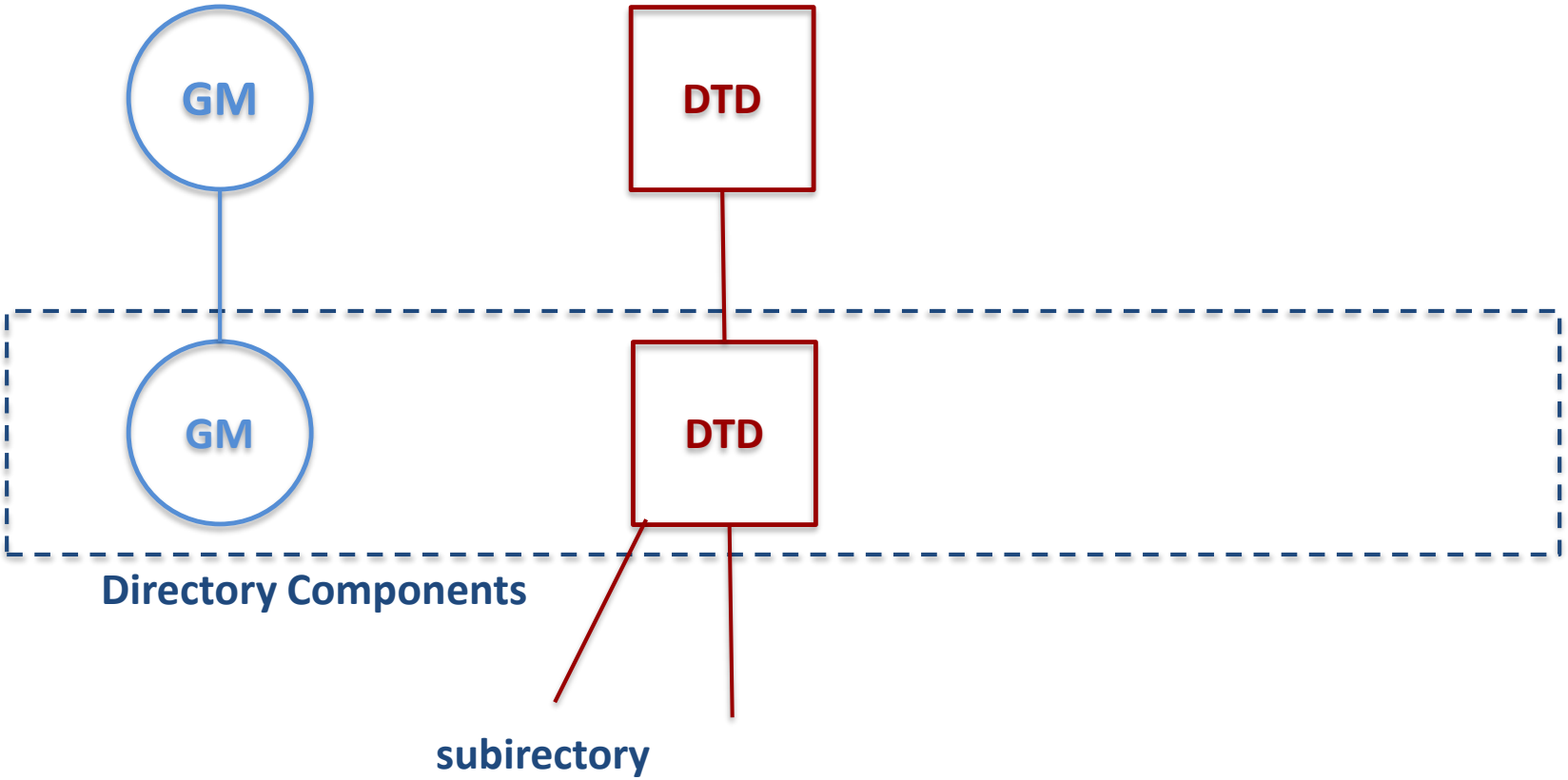
- GM – Huge mseg gives directory tree roadmap

GARNETS Directory Structure

Components of GARNETS Directory

- GM – Huge mseg gives directory tree roadmap
- DTD – Directory Tree DACL
  - Control access to tree from which directories are built
  - ACLs for directory entries

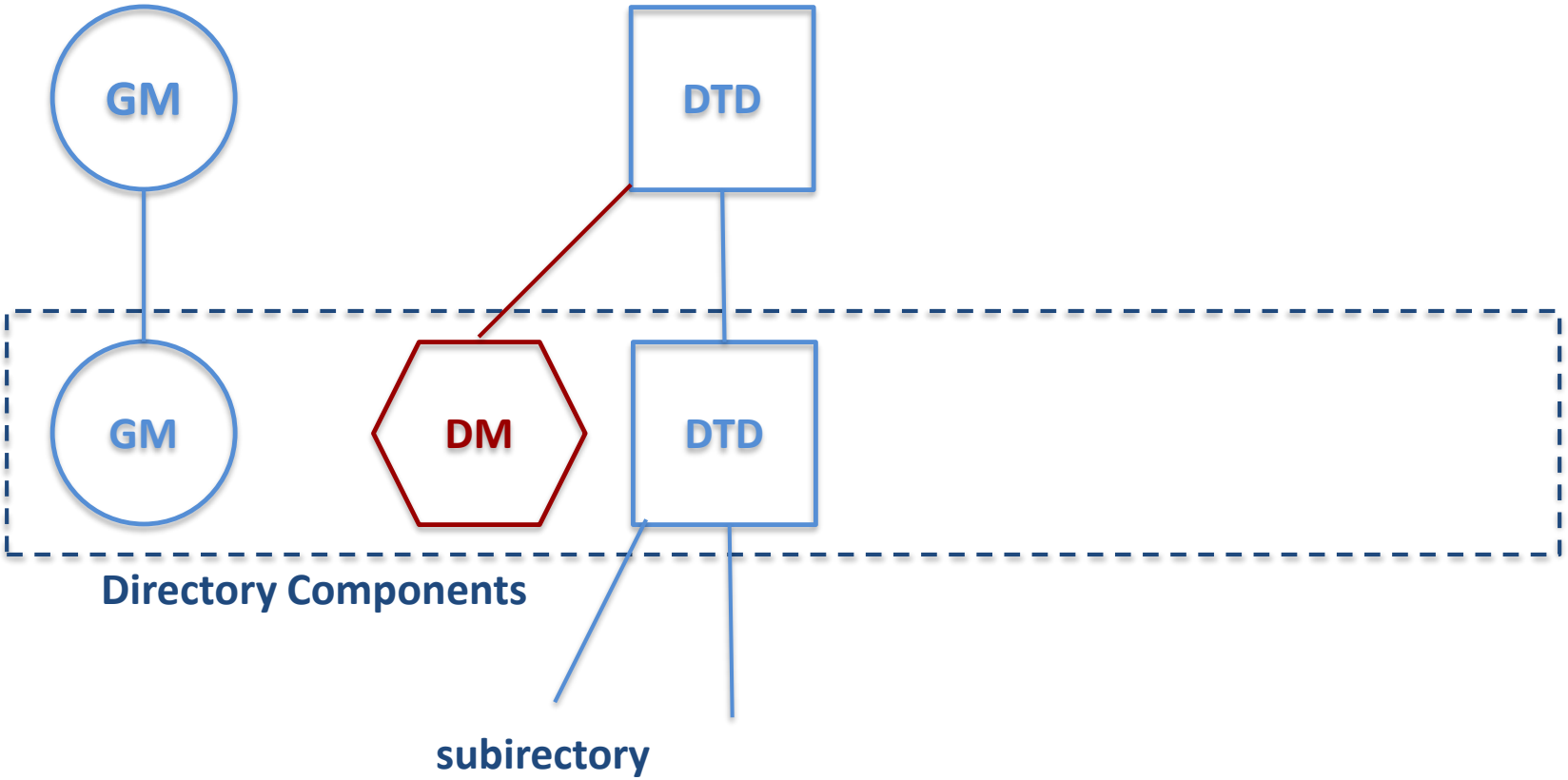
Directory Tree DACL Component



Components of GARNETS Directory

- GM – Huge mseg gives directory tree roadmap
- DTD – Directory Tree DACL
  - Control access to tree from which directories are built
  - ACLs for directory entries
- DM – Directory Multisegment
  - Dynamic data for directory entries

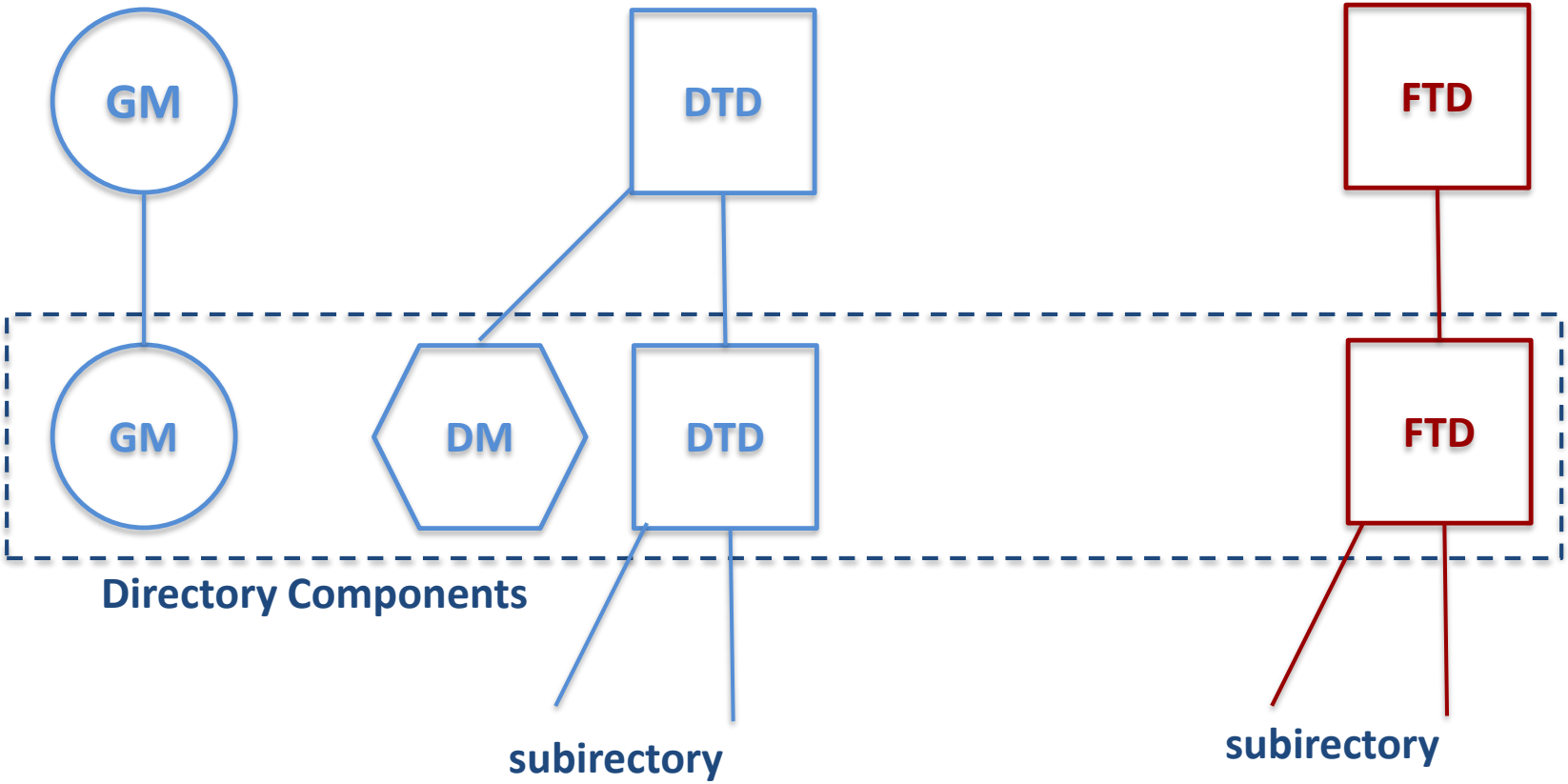
Directory Multisegment Component



Components of GARNETS Directory

- GM – Huge mseg gives directory tree roadmap
- DTD – Directory Tree DACL
  - Control access to tree from which directories are built
  - ACLs for directory entries
- DM – Directory Multisegment
  - Dynamic data for directory entries
- FTD – File Tree DACL
  - Used to extend the tree

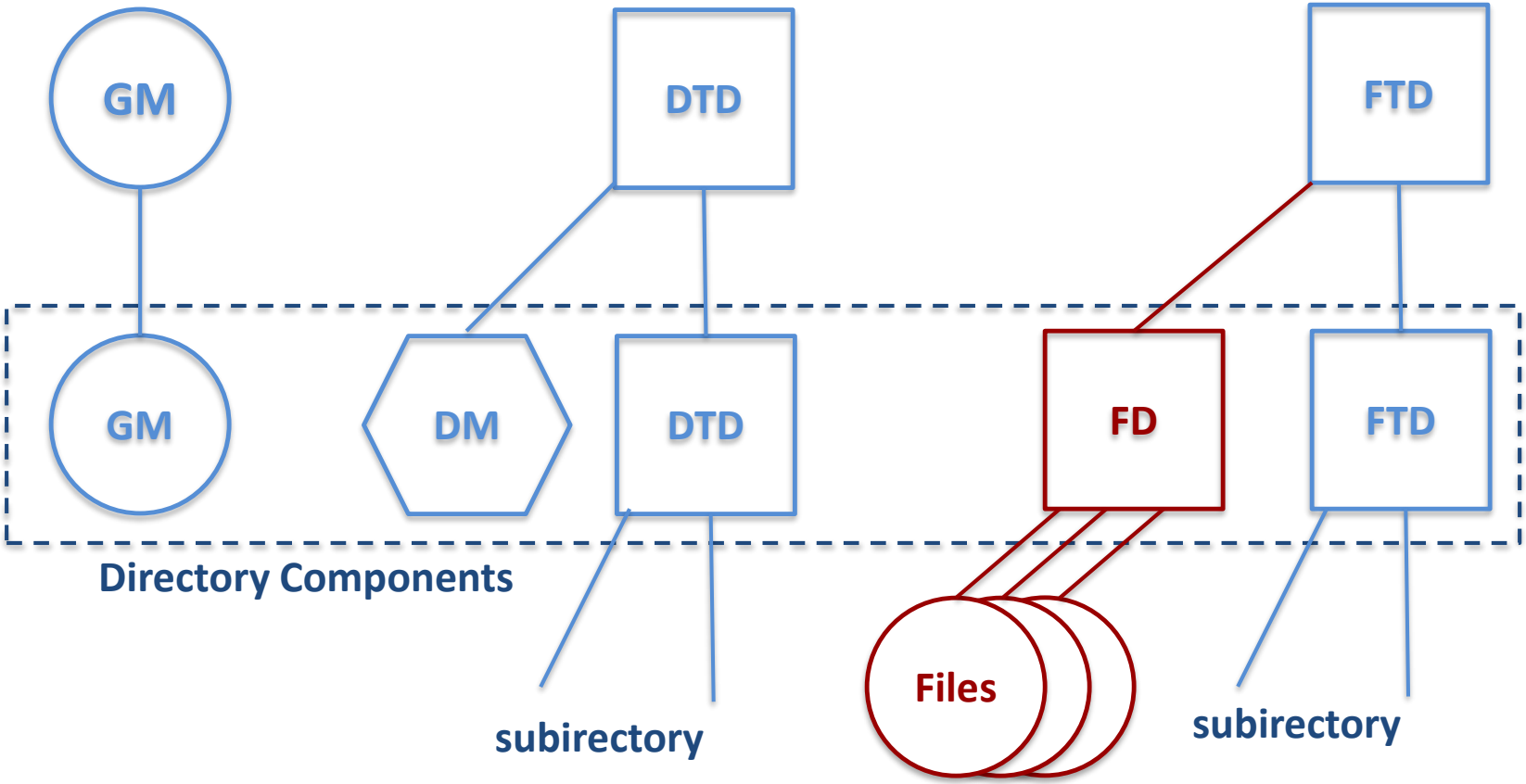
File Tree DACL Component



Components of GARNETS Directory

- GM – Huge mseg gives directory tree roadmap
- DTD – Directory Tree DACL
  - Control access to tree from which directories are built
  - ACLs for directory entries
- DM – Directory Multisegment
  - Dynamic data for directory entries
- FTD – File Tree DACL
  - Used to extend the tree
- FD – File DACL
  - ACLs for file entries in the directory

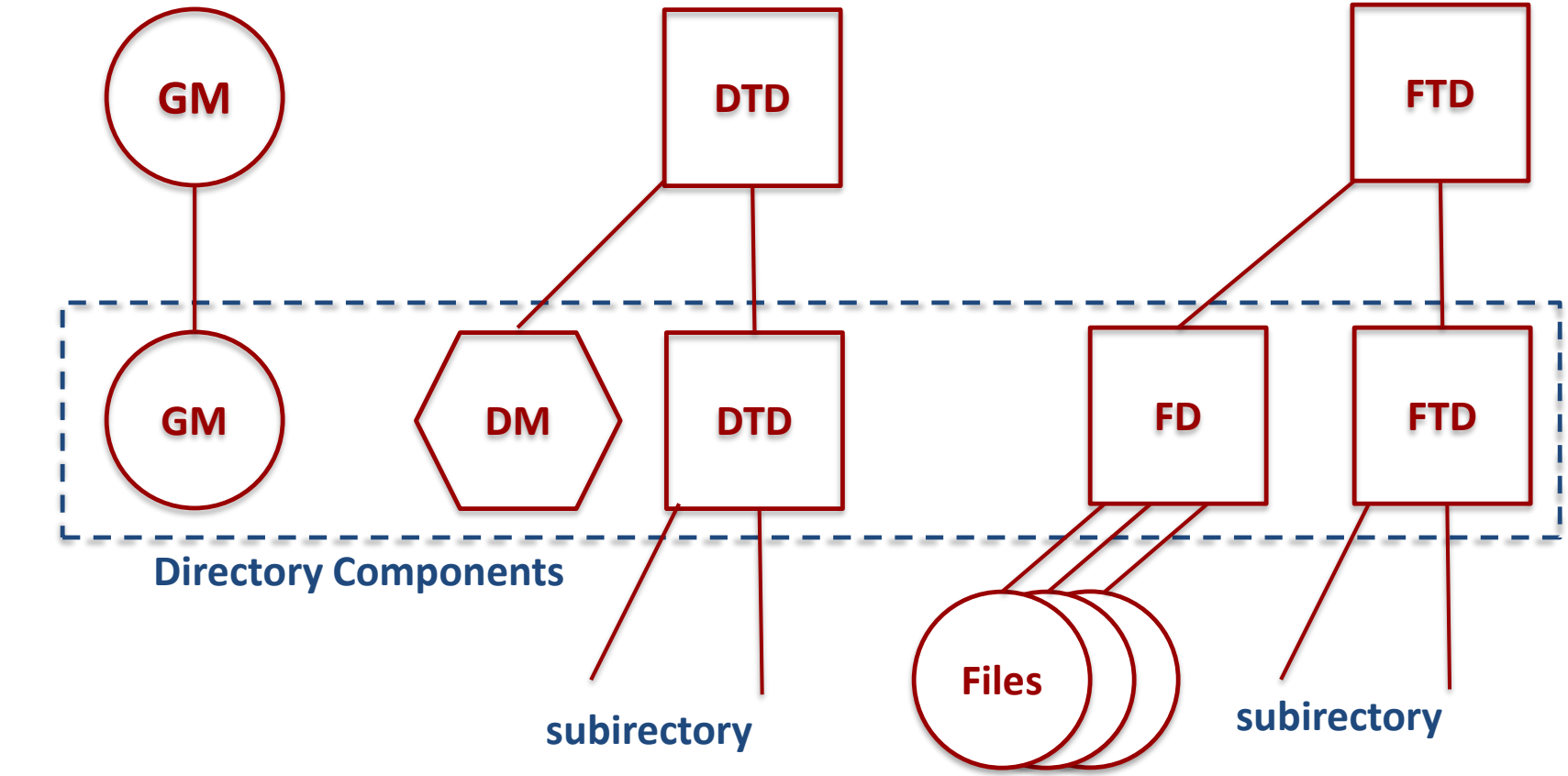
File DACL Component



Components of GARNETS Directory

- GM – Huge mseg gives directory tree roadmap
- DTD – Directory Tree DACL
  - Control access to tree from which directories are built
  - ACLs for directory entries
- DM – Directory Multisegment
  - Dynamic data for directory entries
- FTD – File Tree DACL
  - Used to extend the tree
- FD – File DACL
  - ACLs for file entries in the directory

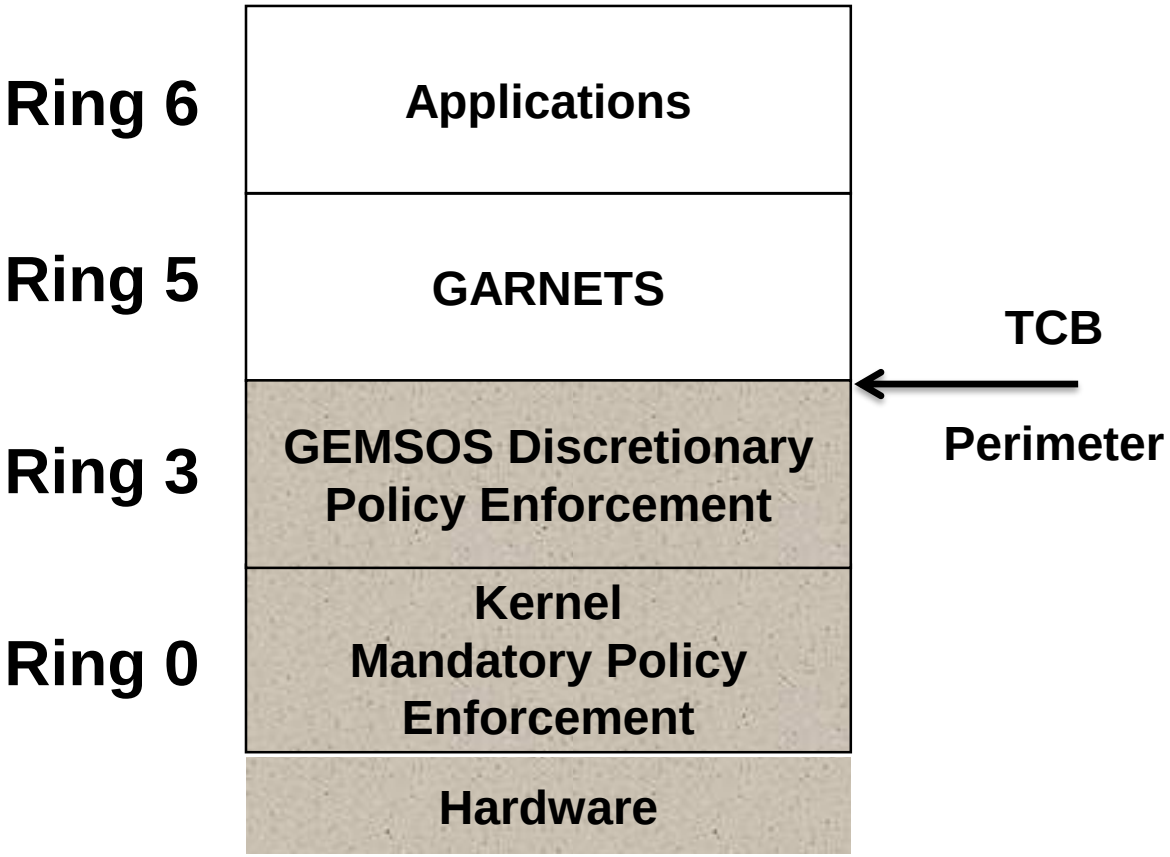
GARNETS Directory Structure



Management of Upgraded Directories

- Initialization is exported to GARNETS interface
  - Must be done by subject at upgraded access class
  - Cannot be done at access class of parent directory
  - For uniformity is same for both normal and upgraded
- Implication for deletion of upgraded directories
  - To meet kernel restriction will require trusted subject
    - Need not have entire range of system's access classes
    - Range encompasses parent and upgraded
- To limit deletion, GARNETS limits creation
  - Users are required to have a "special authorization"
  - Some environments might prohibit user creation

Review of GARNETS Architecture



Named Multisegments (Msegs)

- Msegs are namable directory entries
  - A hierarchically structured collection of segments
  - Single base segment ACL, for all segments in mseg
  - Each segment has an explicit access class
- In contrast to files, not interpretively accessed
  - Segments accessed directly by available hardware
  - Segments are included in process address space
- Msegs used to contain GARNETS internal data
  - Must be protected from less-privileged subjects
  - Uses GEMSOS ring mechanisms to insure integrity
  - Uses DACLs from DTCB to protects internal data

Benefits of Named Msegs

- Avoid unnecessary buffering
  - No per-process file buffer for data
  - Code is not copied from file into executable segments
  - Code is store in right size segments, executed directly
  - Code is not modifiable, so many process can execute
- Promotes sharing of executables
  - Reduced use of real random-access memory
  - Reduced swapping increases performance
- Direct hardware access reduces context switch
- Application can use for databases and libraries
- Highly efficient IPC and synchronization

Single Level Files

- Interpretively accessed as GARNETS interface
  - Applications make calls for file operations
  - Inside GARNETS are created from segments
- GARNETS manages each individual file
  - One ACL is associated with each file
  - Maintains attributes, e.g., time of last modification
  - Time of last read only updated for same level subjects
- Design rejected multilevel files
  - Careful application design eliminates most needs
  - Would create incoherent interface
  - Not clear how to avoid multilevel (trusted) subjects

The Gizillion Problem

- Problem of very large number of access classes
  - Must be addressed for flexible untrusted applications
  - Potential many access classes in underlying TCB
  - GEMSOS: two sets of 16 levels and 96 categories
- TCB minimization limits complex data structures
  - Objective is avoiding elaborate constructs
  - GEMSOS provides **one** base object per access class
  - Each access class must construct its own data
- Handle previously unencountered access class
  - GARNETS subject must create data for applications
  - At minimum OS creates application stack at level

Alternatives for New Access Classes

- GARNETS administrator creates data structures
  - Users requests directory at the new access class
  - New upgraded directory below system low root
  - File system data structures at each new access class
  - Administrator requires access to full range of classes
  - Depends on timely response by administrators
- Create all possible access classes a priori
  - A base directory is always available when needed
  - BUT is untenable to created a gizillion bases
- Trusted process to automate creation process
  - Designers would fail to meet “untrusted” objective
  - Far too complex to meet high assurance

GARNETS Gizillion Problem Strategy

- DTCB creates DACL at first occurrence of class
  - DTCB has function to locate that DACL
  - Has “access class to path” algorithm to base segment
- At first occurrence GARNETS builds a directory
  - Per Access Class (PAC) directory at new class
  - Location for application “home” directory
- GARNETS can then support non-TCB subjects
- TCB tools install GARNETS bootstrapping code
  - Code not located in GARNETS file system.
  - In separate data structures at predefined location

File System Object Naming

- Alias names for objects supported by GARNETS
  - All names must be deleted before object deleted
- Symbolic links are path to target object
  - TCB prevents creation of hard links
  - Can have links to files and named megs
  - Can have links to directories and other links
- Existence of intervening links invisible on access
  - Cycles controlled by number of links traversed in path
- Per Access Class (PAC) links
  - Link has a field for the access class
  - GARNETS finds PAC directory for access class

Leveraging Gizillion Solutions

- Supports use of single-level volumes
  - Single level file systems distributed on volumes
  - Symbolic links permit binding to multilevel structures
- Volumes transparent to application data access
  - Volume access class range implies covert channels
  - Volume range simplifies physical control of media
- GARNETS supports working directories
  - Simplifies naming of subordinate objects
  - Multiple working directory employed for volumes

GARNETS Self-protection

- GARNETS uses rings properly to be effective
  - Applications operate in a less privileged domain
  - Interpretively access objects protected, e.g., files
  - Internal data structures protected from applications
- GARNETS ring brackets
  - Some directories are dedicated to use by GARNETS
  - Range of rings of subjects that will be granted access
  - Apply to all objects in a directory
  - Permanently set when directory is created

Consistency and Concurrency

- File consistency
  - Used to address discontinuities in operation
  - Permit fine-grained robustness selection
- File System Concurrency Control
  - Doesn't insure total ordering of file system operations
  - Each file system object has a version number
- Leverages TCB primitive for atomic updates
  - Avoids conflict with real-time properties
  - Strict two-phase commit for directory components
  - Kernel API can atomically update doubly threaded list

Summary of MLS Implications

- GARNETS file system on high assurance TCB
  - Represents complex general-purpose application
  - Untrusted implementation is MLS context
  - Sufficiently flexible for broad spectrum of uses
- File system managed by single-level subjects
  - Leverage symbolic links
  - Solution to gizillion problem
  - Employable in single-level volume configurations
  - Permits upgraded directories and multilevel msegs
- GARNET protects itself and its data structures
  - Exploits rings and DACLs from high assurance TCB

INF523

Subversion Case Study NFS

Professor Clifford Neuman

Network File Service (NFS) Security

- Case study of NFS subversion demonstration
  - Running example by US Navy masters student
  - Emory A. Anderson, III, for Prof Cynthia Irvine (NPS)
  - Shown to Richard Clarke, “first cybersecurity czar”
- First, consider security implications for system
  - How deeply rooted are adverse consequences
- Second, explore applicability to other systems
  - Address whether attack approach is limited to NFS
  - Briefly examine Anderson SSL subversion design
- Follow on – Later NFS case study of mitigation
  - Compare to Anderson recommended solution

Likely Tool of Professional Attacker

- Subversion is technique of choice [And 1.D]
  - Professional distinguished from amateur
- A primary objective is avoiding detection
  - Amateur often motivated by desire for notoriety
- Professional often well-funded
  - Resources to research and test in closed environment
  - Amateur tends to numerous attempts on live target
  - Flawless execution reduces risk of detection
- Coordinated attacks are mark of a professional
- Professional will invest and be patient to use
  - Subverter is likely different than attacker

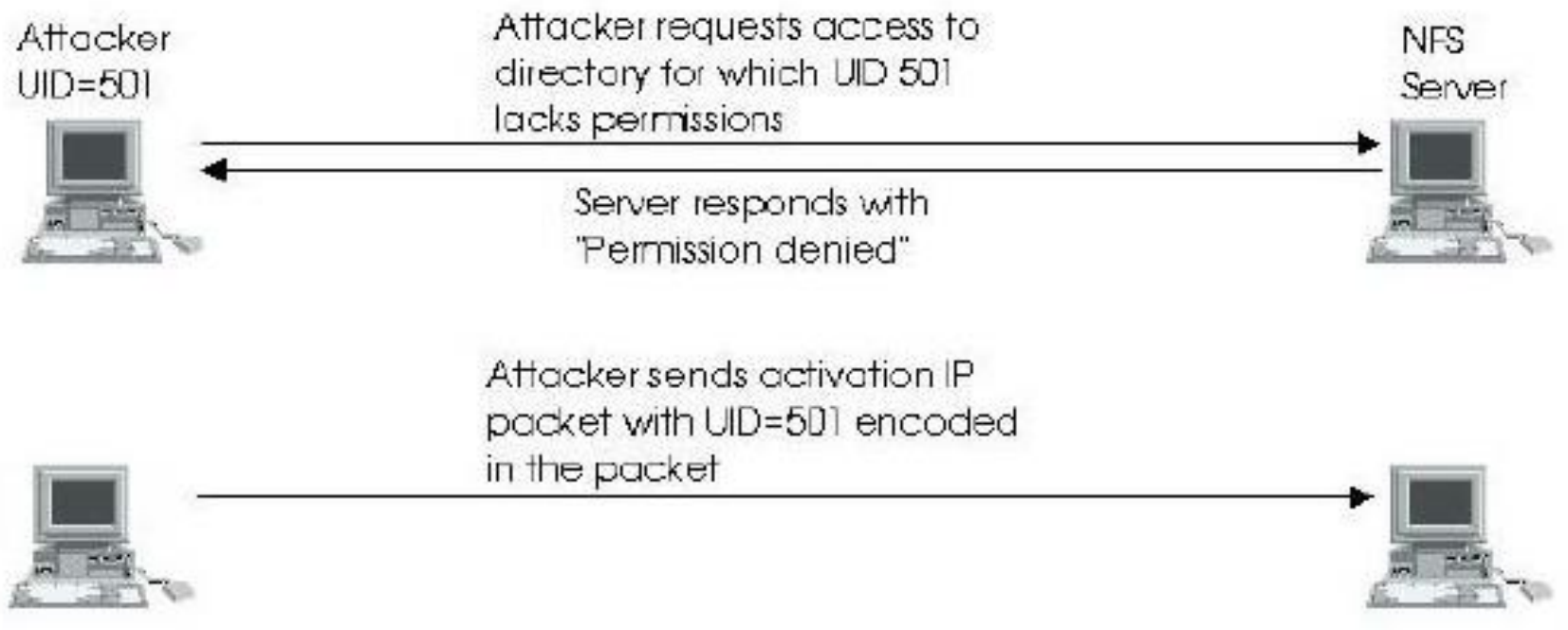
Demonstration of Subversion

- Obfuscation of artifice not given serious attention
  - Would be of utmost importance to professional attack
- Subversion can occur multiple points in lifecycle
- Selected distribution phase for demonstration
  - Driven by limited resources and access of student
  - Facilitated by NFS on open source Linux system
  - Representative of attacker mirror site opportunities
- Closed source not daunting for professional
  - May involve reverse engineering application
  - Might create a binary patch to insert in binaries
  - Entirely within anticipated professional resources

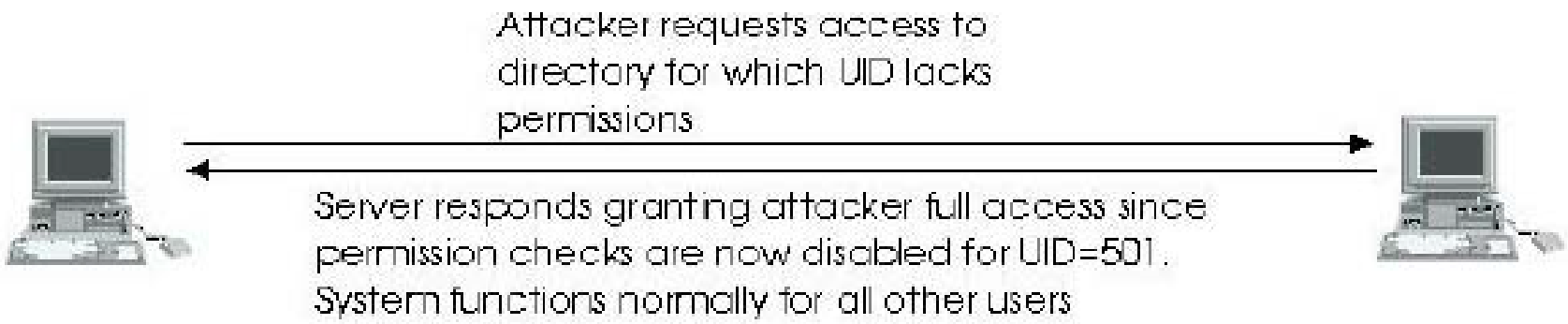
Choice of NFS as Suitable Application

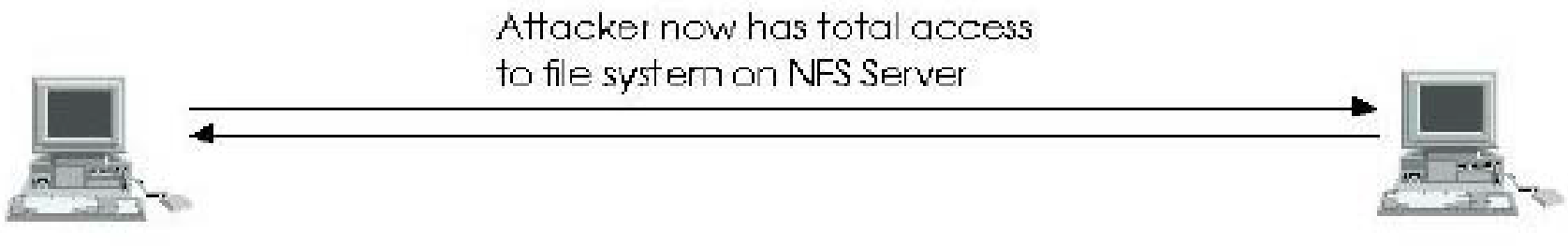
- For impact, need readily apparent significance
  - NFS is application familiar to typical IT user
  - Users understand notion of need to protect data
- Activation needs to be straightforward
  - Network interface chosen for ease of explanation
  - Internet technology is widely used
- Choose to have remote activation
  - Representative of low risk for attacker
  - Also supports local activation, e.g., via loopback
  - Trigger is a malformed Internet packet
- **Study of subversion method benefits student**

Case System and Activate the Artifice

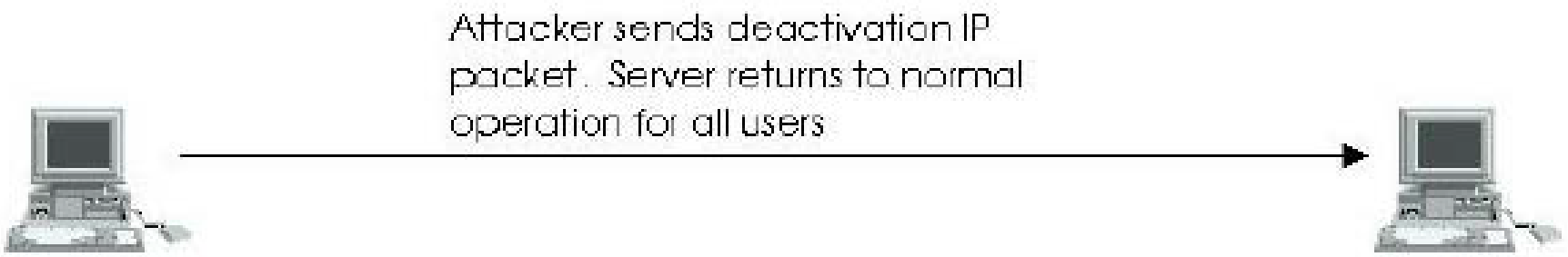


Attacker Uses Artifice for NFS Access





End Session by Deactivating Artifice



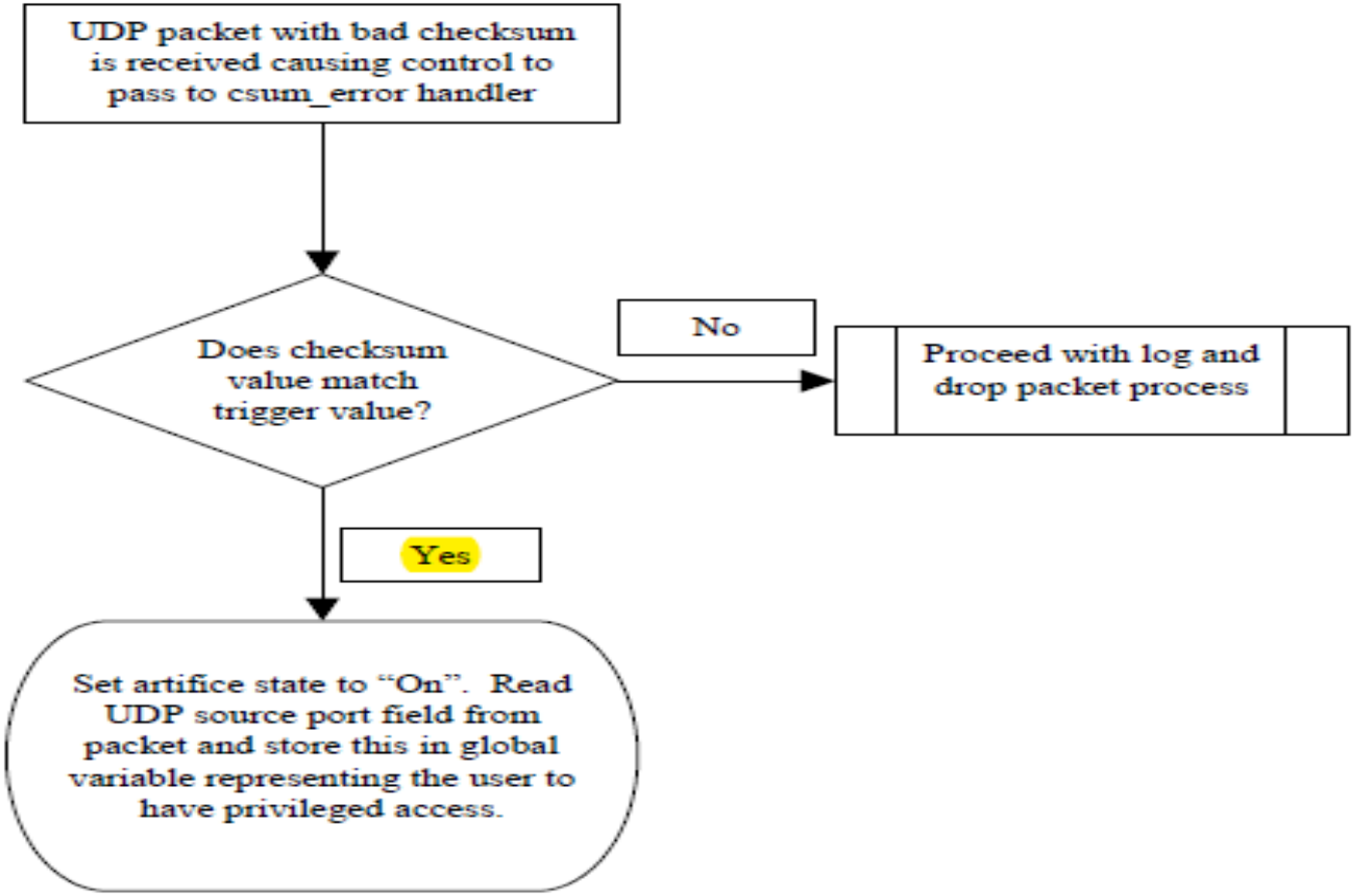
Design Properties of NFS Artifice

- Purpose of artifice to bypass file permissions
  - Bypass check for a specified user at will
  - Then re-enable the normal system operation
- Exhibits all the characteristics of subversion
  - Exception was no attempt for hide or obfuscate
- Artifice is small – eleven C statements
  - Small in relation to millions LOC in Linux kernel
  - Unlikely to be notice by those in Linux development
- Can be activated and deactivated
  - Further complicates attempts to discover existence
- Does not depend on activities of a system user

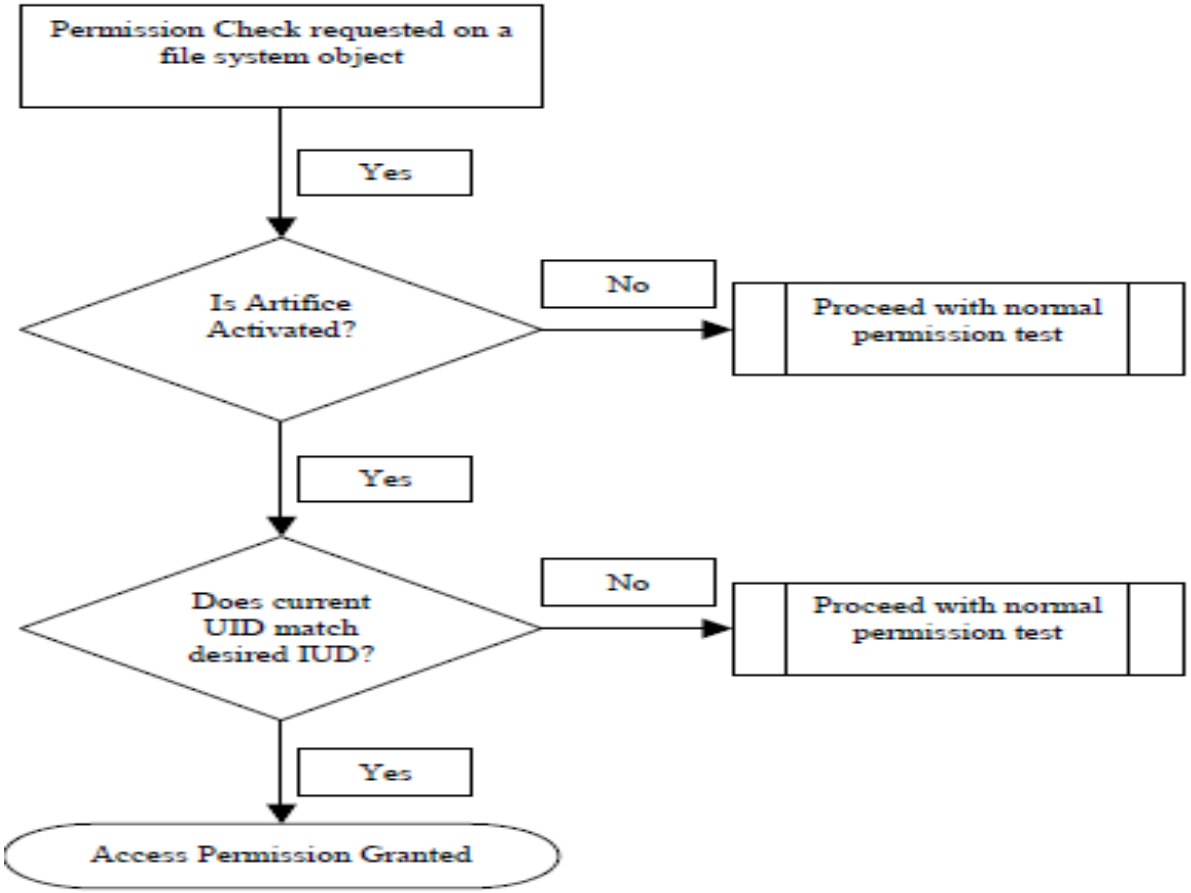
Artifice Functions

- Composed of two parts in two unrelated areas
- Subvert a portion of kernel that receives packets
  - Recognize a packet with distinguishing characteristics
  - Activation based on trigger known only to subverter
  - Extends normal check for packet checksum
- Activation recorded in global variable in kernel
- Subverts Linux file system permission checks
  - Check global kernel variable to see if activated
  - Grants attacker access to any file in the system
  - Bypass behavior limited to specified user ID
  - System functions normally for all other users

Artifice Activation



Subverted File Permission Checks



Separate Design of SSL Subversion

- Secure Sockets Layer (SSL) widespread use
  - Secure communications between client and server
  - Client and server negotiate session keys
  - Encrypt traffic using symmetric encryption algorithm
- Options available to attacker for subversion
  - Duplicate all communications and send to attacker
  - Weaken key generation mechanism – limit entropy
  - Simply send the session keys out to the attacker
- Advantages of exfiltrating session keys
  - Attacker is passive and maintains anonymity.
  - Subverting either client or server gives total acess

NFS Subversion Technical Conclusions

- Practice for showing security inadequate at best
  - Penetration tests and add-on third party products
  - Layered defenses and security patches irrational
- Bad defense more dangerous than poor security
  - Leads to flawed belief system has adequate security
  - Can increase risk by more dependence on defense
- Have technology to provide appropriate security
  - Evaluation criteria tried and tested
  - These approaches have fallen into disfavor
- The need to address subversion is increasing
  - Threat sources multiplying and reliance increasing

NFS Subversion System Decisions

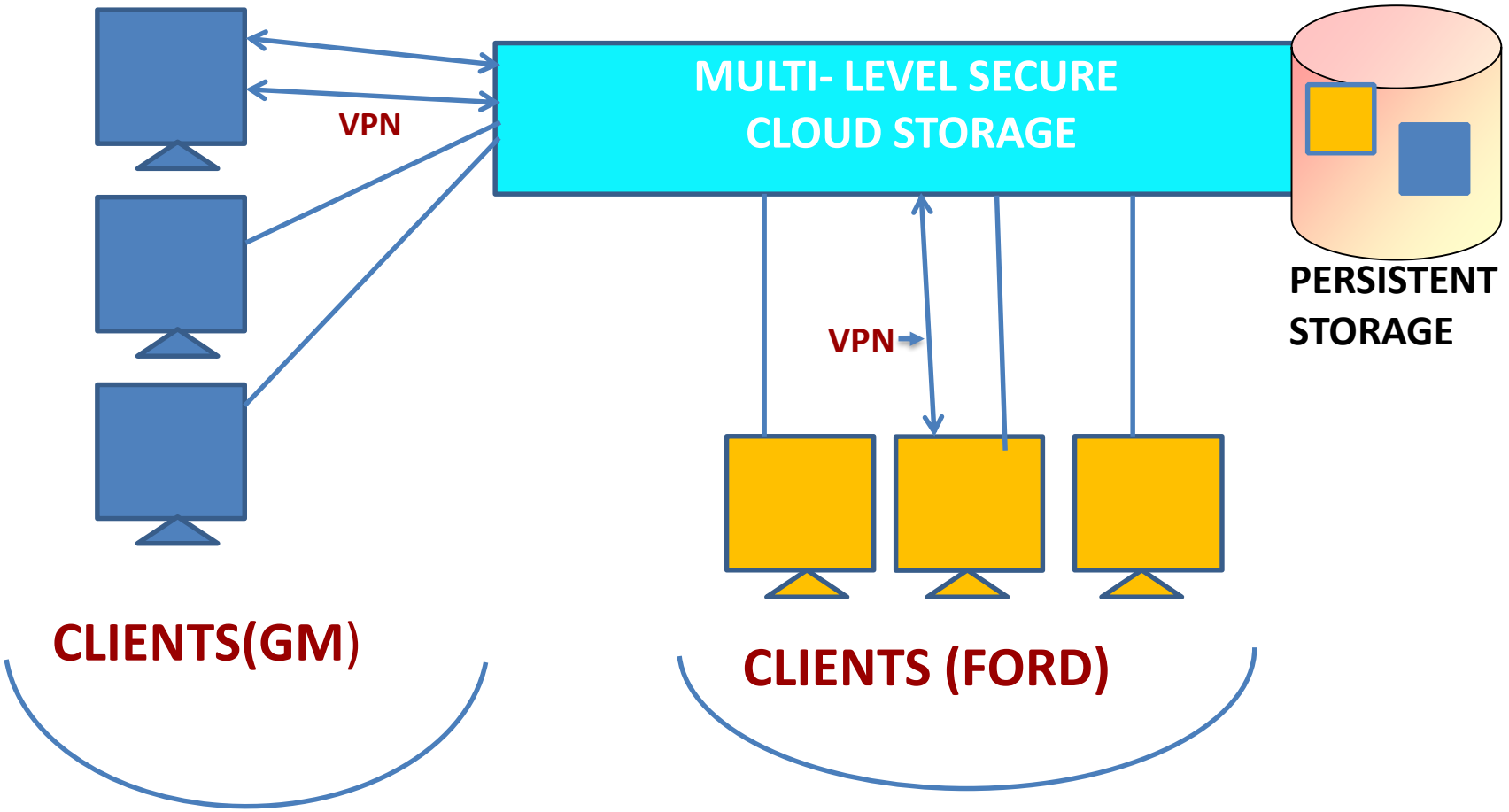
- Must address subversion for justification of trust
  - Irresponsible not to consider when deploying systems
  - Otherwise flawed belief system security is adequate
- Nurture a vast industry with add-on applications
  - Huge drain on resources for little or no assurance
- Objective of demonstration to raise awareness
  - Enable decision maker to understand the problem
    - Need to understand motive, means and opportunity
    - Consider subversion practicality and consequences
  - Make decision makers aware of proven technology
    - Verifiable protection technology applied successfully
- **Security professionals have a responsibility**

Recall Study Goals for NFS Subversion

- First, consider security implications for system
  - How deeply rooted are adverse consequences
- Second, explore applicability to other systems
  - Address whether attack approach is limited to NFS
  - Briefly examine Anderson SSL subversion design
- Next – NFS case study of mitigation
  - Compare to Anderson recommended solution

- **What else can be learned from the demo?**

Notional Cloud Storage Security



MLS File Sharing Server for Cloud

- Cloud storage service
  - Specific type of cloud computing
  - Managed resource is storage
- Needs security as good as enterprise
  - Typically replaces services of enterprise environment
  - Many of the same vulnerability as self-managed
  - Additional vulnerabilities specific to the cloud
- Current solutions are completely ineffective
  - Essential problem is construct of shared infrastructure
  - Built on low-assurance commodity technology
- Highly vulnerable to software subversion

Present-day Vulnerability Examples

Apple pushes security and privacy credentials after iCloud hack

Dark cloud: Study finds security risks in virtualization

Hackers penetrated Nasdaq computer network

Security Requirements of cloud

- Three primary cloud security requirements
  - Controlled sharing of information
  - Cloud isolation
  - High Assurance

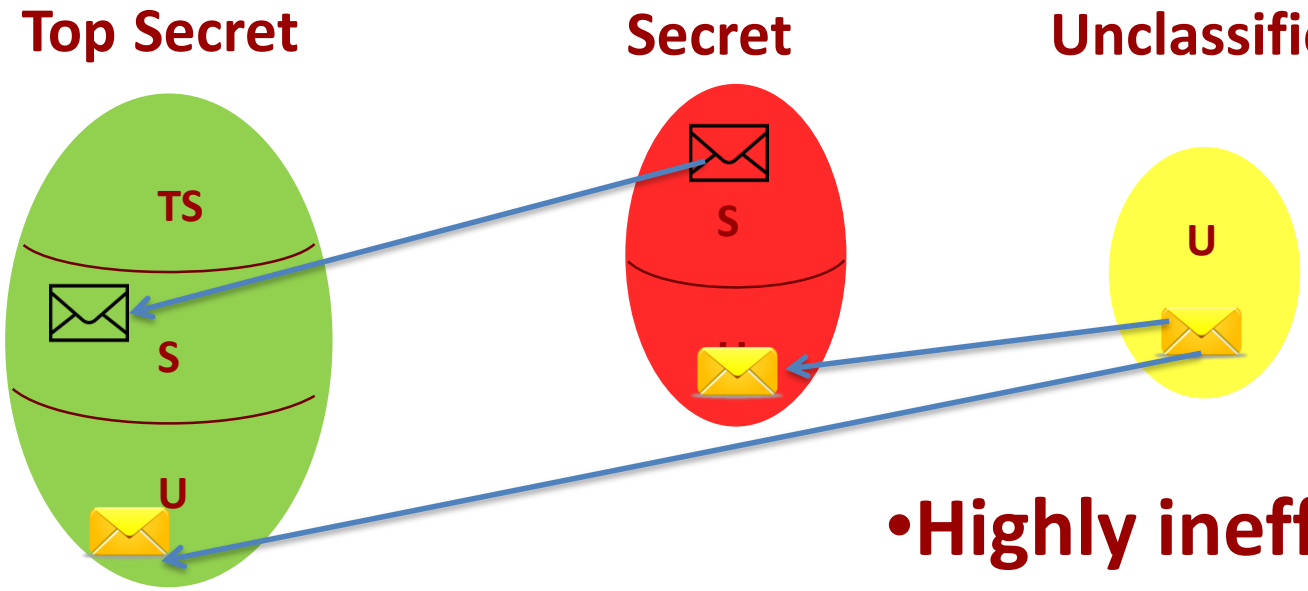
Trap Door Subversion Vulnerability

- Malicious code in platform
  - Software, e.g., operating system, drivers, tools
  - Hardware/firmware, e.g., BIOS in PROM
  - Artifice can be embedded any time during lifecycle
  - Adversary chooses time of activation
- Can be remotely activated/deactivated
  - Unique “key” or trigger known only to attacker
  - Needs no (even unwitting) victim use or cooperation
- Efficacy and Effectiveness Demonstrated
  - Exploitable by malicious applications, e.g., Trojans
  - Long-term, high potential future benefit to adversary
  - Testing not at all a practical way to detect

Alternatives for Controlled Sharing

- Three ways controlled sharing can be facilitated:
- Massive copies of data from all lower levels
  - High assurance one-way flow of information
  - Light diode interface uses physics for high assurance
- File Caching (Local Active Copy)
  - Retain at high level only actually used lower data
  - No way to securely make requests for lower data
  - Security requires manual intervention
- High assurance segmented virtual memory

Massive Copies Approach



- **Highly inefficient!**
- **Does not scale!**

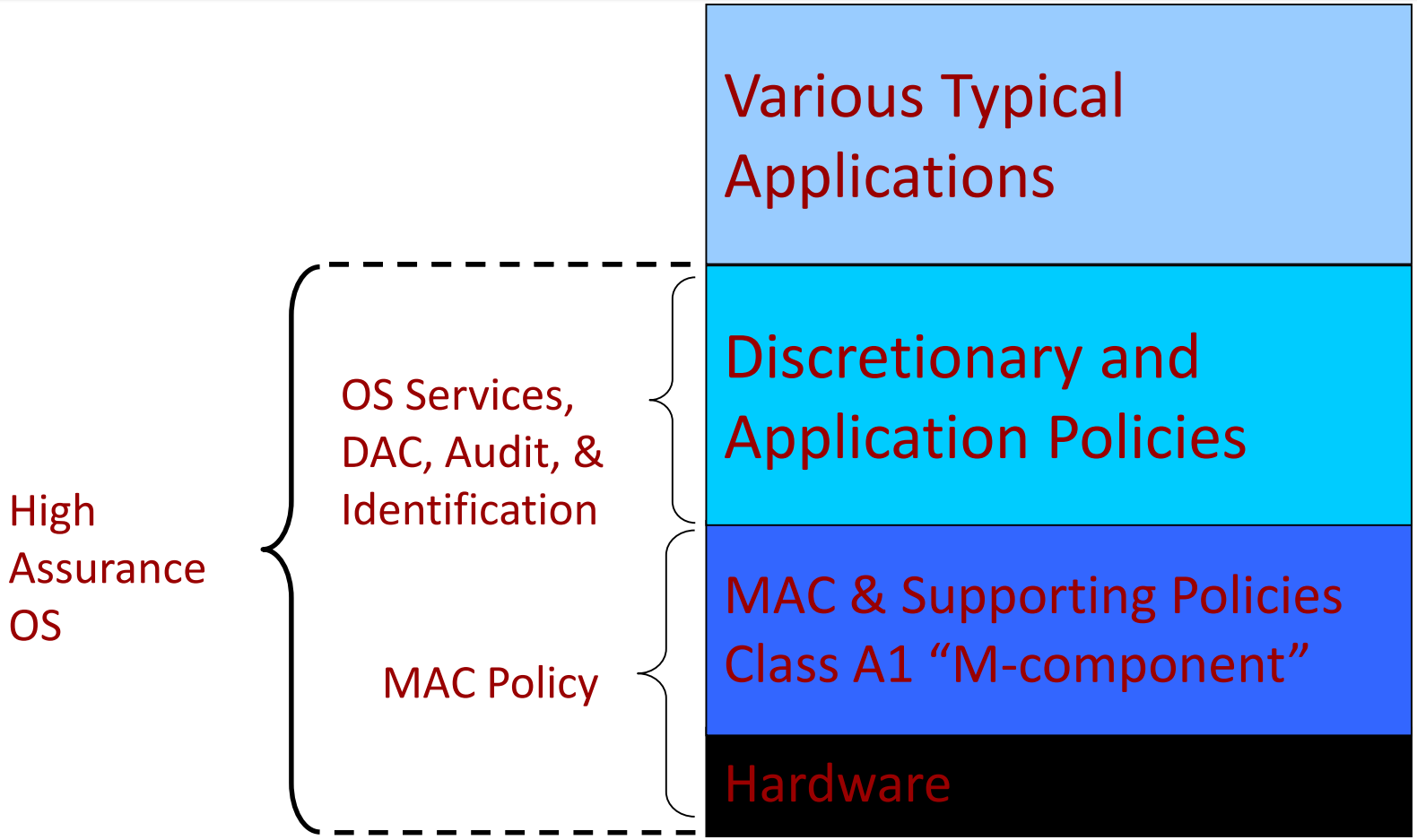
Computer Utility: Predecessor to Cloud

- Computer Utility made security a priority
- Anticipated present day cloud environment
  - Controlled sharing was an integral requirement
  - Incorporated MAC policy
    - Evident from commercial Multics product.
  - Consistent with high assurance
    - Evident from the BLP Multics interpretation.
- Didn't gain widespread acceptance
  - Before Internet enabled today's cloud computing

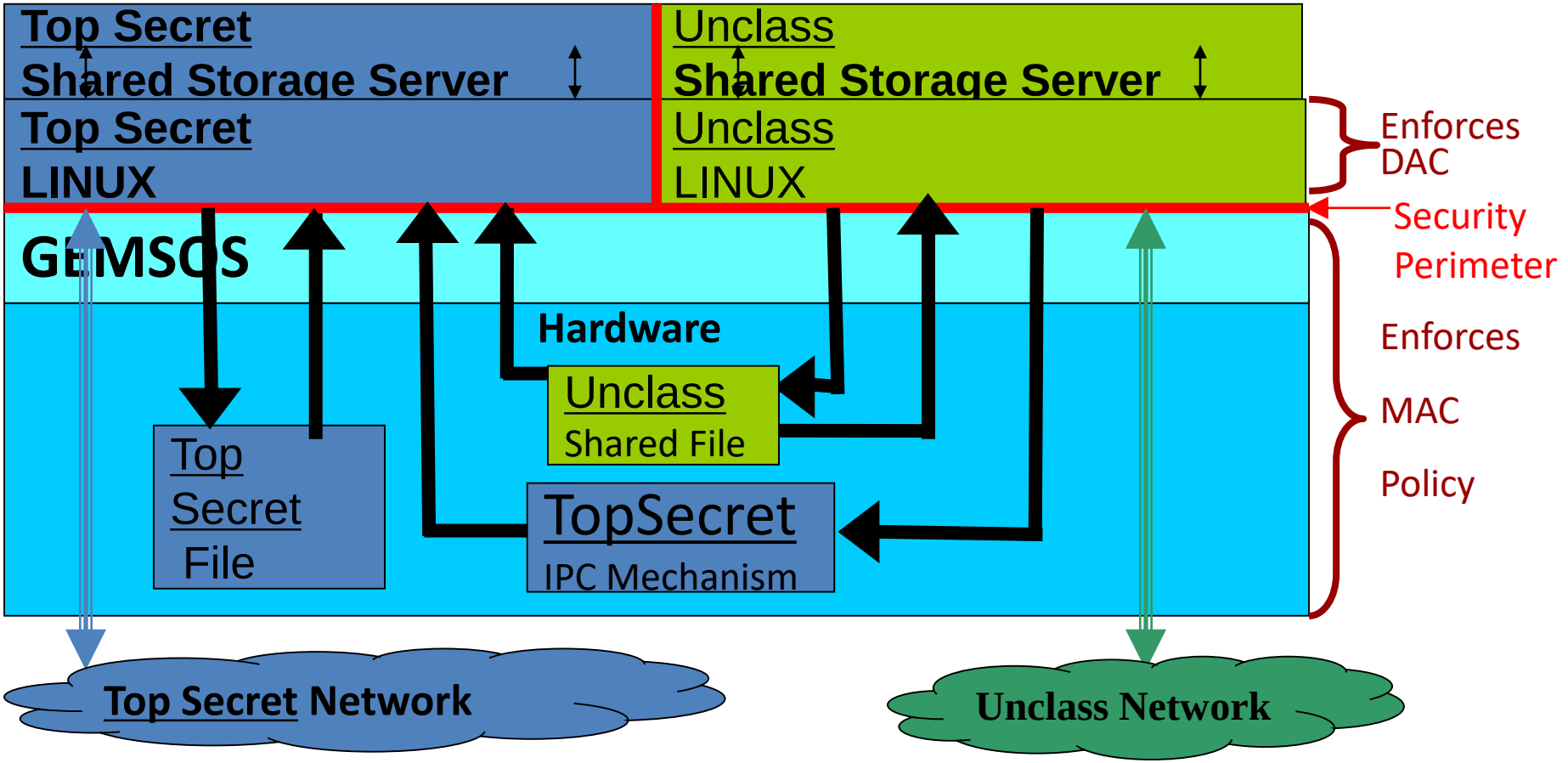
Basis to Consolidate Networks

- Foundation: Trusted Computing Base (TCB)
  - The totality of protection mechanisms
  - within a computer system
  - including hardware, firmware, and software
  - Responsible for enforcing a security policy
- The security perimeter is TCB boundary
  - Software within the TCB is trustworthy
  - Software outside the TCB is untrusted
- Mature technology -- 30 years experience
  - Derived from and used for commercial products
- Key choice is assurance: low to very high

TCB Subsets –"DAC on MAC"



Use Platform for Controlled **Sharing**



Motivation to Address Cloud Security

- Cloud storage flexible, cost-effective
- How to implement in multi-level environment?
  - Duplicate for each level? Loses advantages.
- Tempting target for attackers
  - Can increase privilege
- Want high-assurance, MLS solution
- Want cross-domain sharing (CDS)
  - Share resources are core cloud value proposition
  - Controlled sharing of up to date information
- Solution: MLS cloud network file service
  - Based on high-assurance, evaluated Class A1 TCB

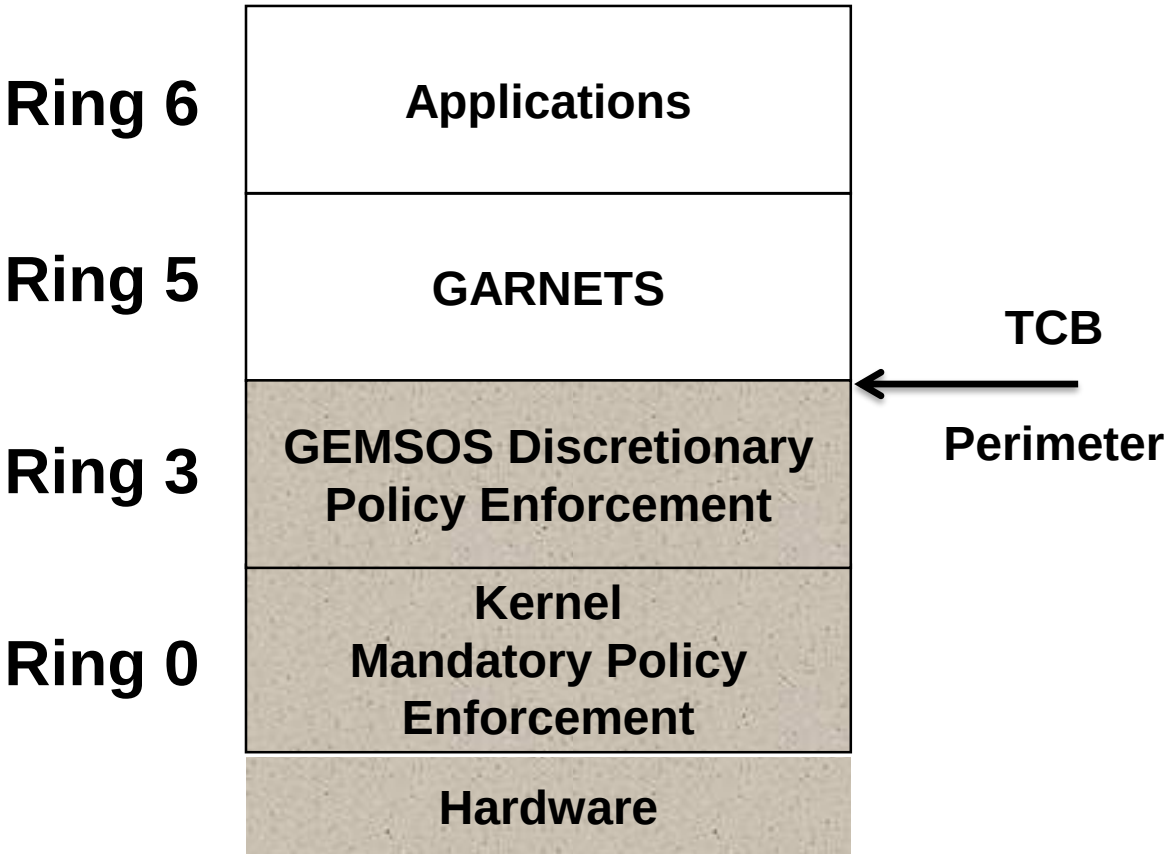
Cloud Storage Security Challenges

- Migrate storage from local security domain
  - Lose protection of “air gap” between domains
  - Dramatically expands opportunity for adversaries
- Common security approaches can’t work
  - Power of insidious software subversion attack tools
  - Exploding “cloud computing” amplifies impact & risk
- Proven reference validation mechanism (RVM)
  - Systematically codified as TCSEC “Class A1”
- Need architecture that leverages trusted RVM
  - Use verifiable (i.e., Class A1) trusted computing base
  - Want high cloud compatibility, e.g., standard NFS

Current Cloud Technology is Vulnerable

- Typically on commodity low-assurance platforms
- NetApp StorageGRID Systems
  - On Linux servers or VMware hypervisors
- OnApp Storage cloud
  - On various hypervisors, including Xen
- Amazon Elastic Compute Cloud
  - On Xen hypervisor to isolate guest virtual machines
- Xen is representative example of low-assurance
  - So-called TCB includes untrustworthy Linux, drivers
  - Largely unconstrained opportunities for subversion
- NSA said system on partition kernel too complex

Review of GARNETS Architecture

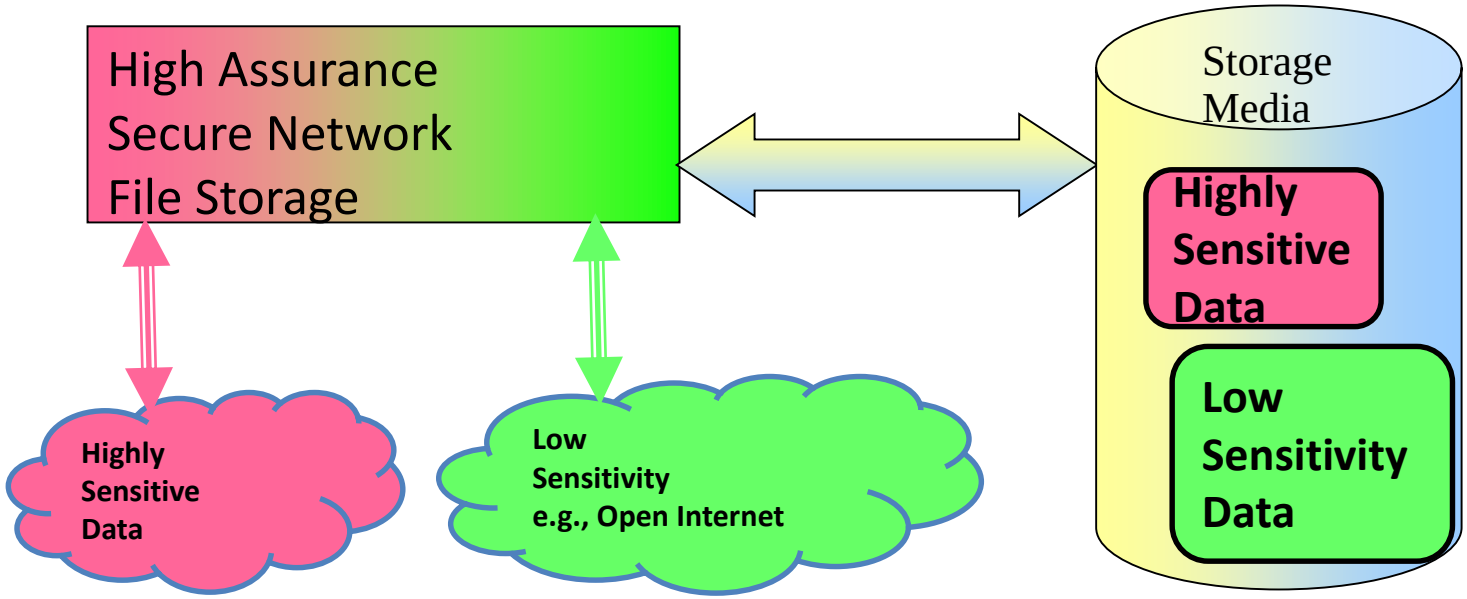


Build on MLS File System Foundation

- GARNETS MLS File is untrusted application
  - In a layer, in a separate ring, above the TCB
  - Creates file management that uses TCB objects
- MLS file system acts like standard file system
  - Spans multiple domains protected by the TCB
  - Creates one logical name space for all domains
- Run GARNETS instances at multiple domains
  - Means no massive copies needed for sharing
  - Can read both directories and files of lower domains
- For cloud storage need to add network interface
  - Multiple network interfaces for multiple domains

Target Secure Cloud Storage

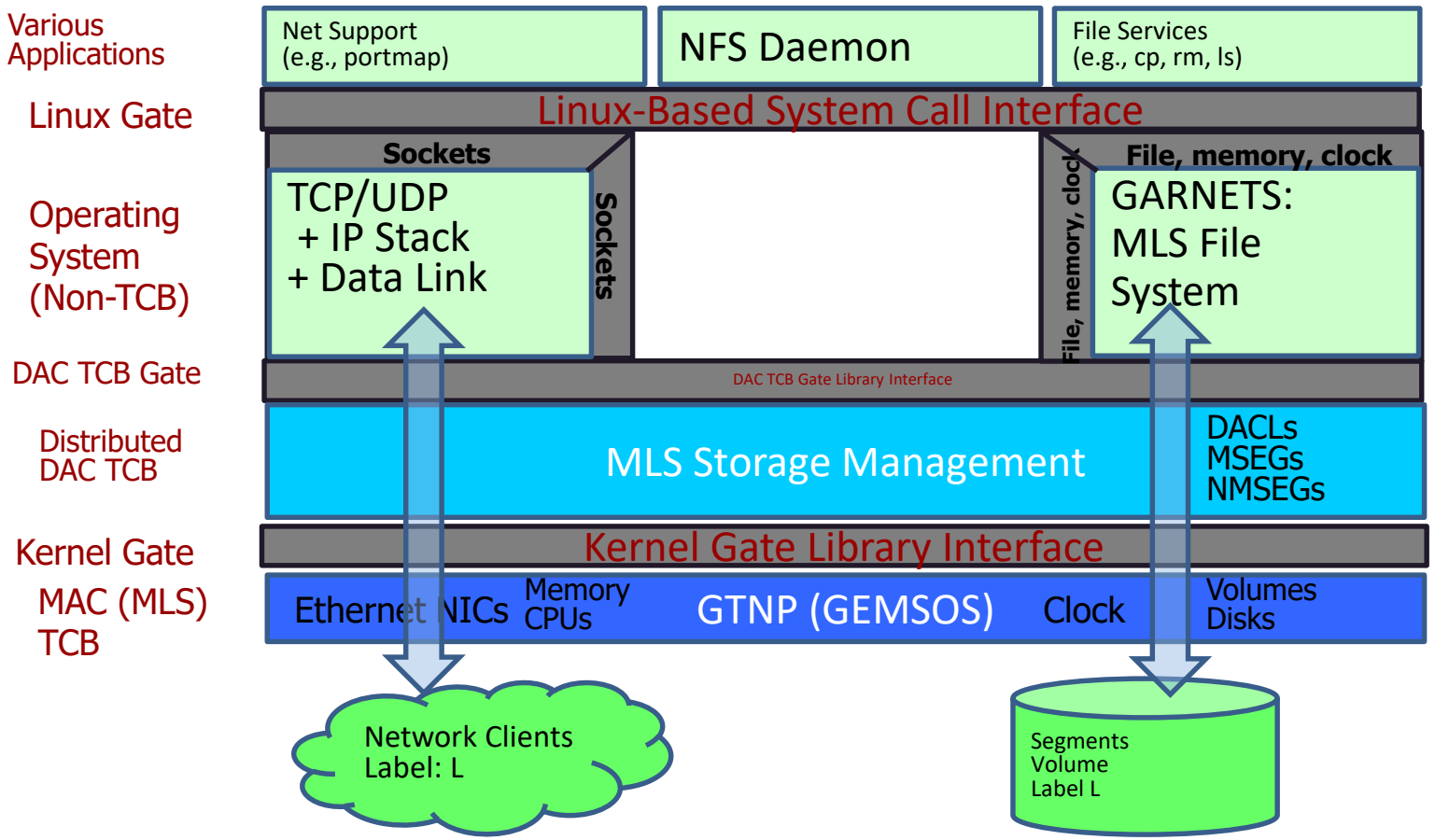
- Operates like standard network file storage
- BUT, verifiable security for
  - MAC separation of security domains



Run NFS on Top of MLS File System

- **System** challenge is a mostly compatible NFS
  - Accept the few intrinsic limitations, e.g., time last read
- Ease of porting NFS wants familiar OS interface
  - Requires creating “compatible OS” [GAS 10.7.2]
  - Demonstration chose POSIX style interface
  - Intended to support Linux source code compatibility
- In contrast, **GARNETS is not compatible**
  - Have to create a Linux system call interface
- Clients access files on server through NSF calls
  - Clients are untrusted
  - Any client using standard NSF protocol can access

NSF Low Domain S/W Instantiation



For MLS Run Multiple NSF Instances

- Need a separate instance for each level
  - Is the only way NFS can be untrusted software
- Clouds often use virtual machine monitor (VMM)
  - Have noted low-assurance virtual machine problem
- Secure NFS demo leverages GEMSOS VMM
  - FER from NSA describes trusted MLS virtualization
  - Is NOT Type I virtualization, i.e,, cannot run binary
  - Each NFS instance runs in its own virtual machine
- MLS from TCB cannot be bypassed by VMM
  - Can be configured for typical isolation
  - In contrast to most VMMs, has controlled sharing

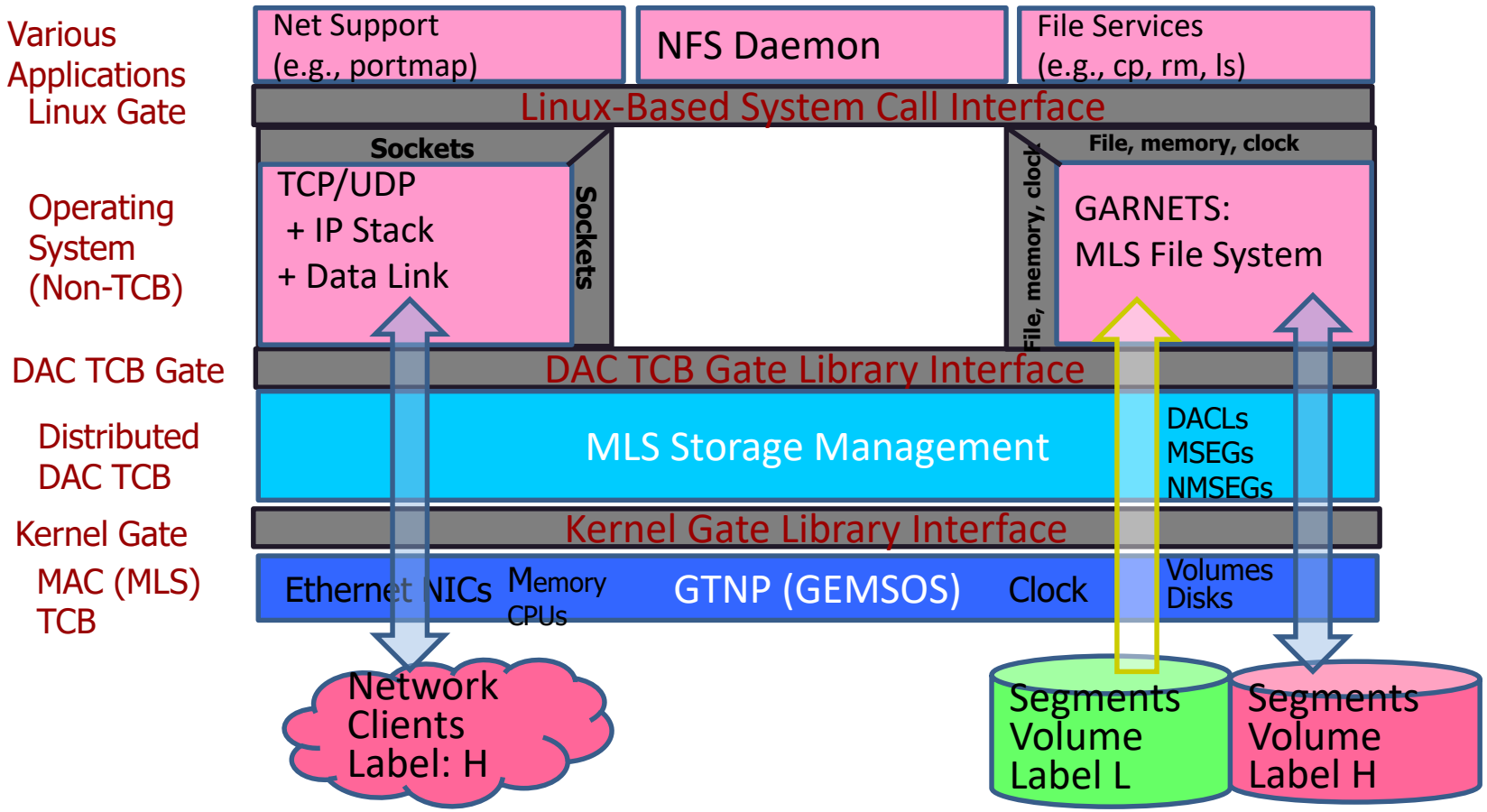
FER References GTNP VMM

- Sec 2.3, para 2:
  - “The GTNP is intended to support "internal subjects"
  - Virtual machines per Section 1.3.2.2.2 of the TNI.
- Sec 2.3.1, page 12:
  - Implementing A-components as virtual machines
  - Layer between M-NTCB and untrusted VM subjects
- Sec 2.3.1, page 12:
  - VM on top of VMM provided by M-component
- Sect 2.3.1.2 Virtual Machine Interface,
  - Way to compose other components with GTNP
- Sec 4.2.1.8.2.1, para 2: VM supports users

More FER References GTNP VMM

- Sec 4.2.2, para 1:
  - Gate with hardware provide a VMM base
- Sec 9.2, page 181:
  - Rings support VMs the NTCB MAC partition.
- Sect10.3, subpara 2:
  - Multilevel device built as a VM on GNTP VMM
- Sect 10.6, para 1:
  - Implement other network partitions as VM
- Appendix C, EPL Entry, page C-2:
  - GTNP supports a virtual machine monitor interface

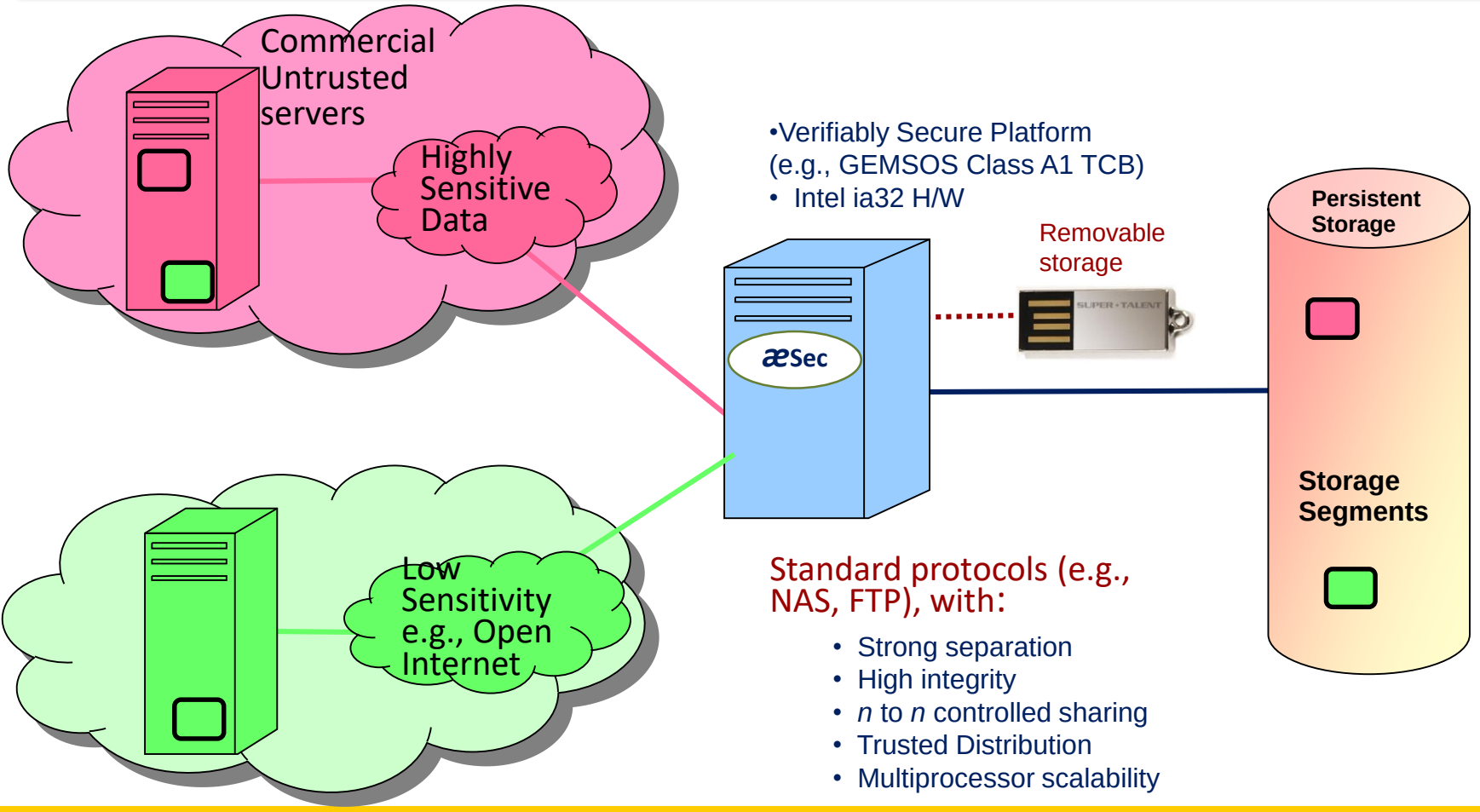
NSF High Domain S/W Instantiation



Can Support Extended Cloud Services

- Usually have metadata server for cloud services
  - Maps object names to actual locations in the cloud
- Single level servers in the network can access
  - May use FTP to transfer files to/from MLS file server
  - Applications NAS protocols can access it files
- BEWARE of sloppy system security engineering
  - Can't initiate information transfers **between** domains
  - TNI provide the network engineering framework
- Separate NIC per domain does not scale well
  - Next look at how to use a single hardware interface
  - Need equivalent of a TCSEC/TNI multilevel device

Security for Untrusted Servers



Cloud Controlled Sharing Summary

- Key is Mandatory Access Control (MAC)
  - Control isolation & sharing between security domains
- Defining properties: global and persistent
  - Control information flow (confidentiality)
    - Prevents malicious information exfiltration
  - Control contaminated modification (integrity)
  - Sound mathematical foundation
- Implement with distinct access class labels
  - Label domain of information with access class
  - User has authorized access classes, i.e., domains
- Supports MAC, viz., multilevel security (MLS)

INF527:
Secure System Engineering
MLS Cloud Storage

Professor Clifford Neuman

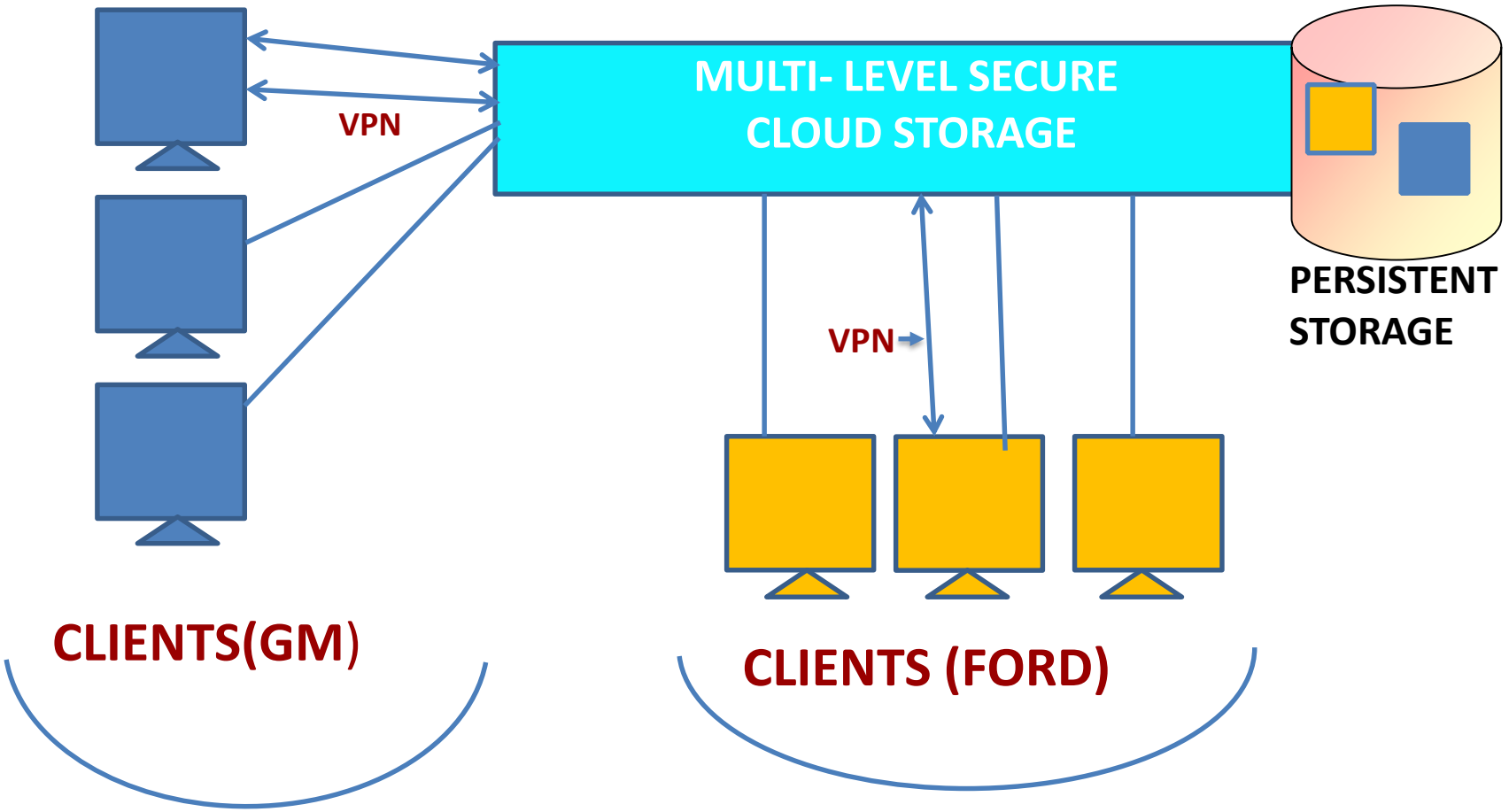
Lecture 13 continued

Recall Study Goals for NFS Subversion

- First, consider security implications for system
  - How deeply rooted are adverse consequences
- Second, explore applicability to other systems
  - Address whether attack approach is limited to NFS
  - Briefly examine Anderson SSL subversion design
- Next – NFS case study of mitigation
  - Compare to Anderson recommended solution

- **What else can be learned from the demo?**

Notional Cloud Storage Security



MLS File Sharing Server for Cloud

- Cloud storage service
  - Specific type of cloud computing
  - Managed resource is storage
- Needs security as good as enterprise
  - Typically replaces services of enterprise environment
  - Many of the same vulnerability as self-managed
  - Additional vulnerabilities specific to the cloud
- Current solutions are completely ineffective
  - Essential problem is construct of shared infrastructure
  - Built on low-assurance commodity technology
- Highly vulnerable to software subversion

Present-day Vulnerability Examples

Apple pushes security and privacy credentials after iCloud hack

Dark cloud: Study finds security risks in virtualization

Hackers penetrated Nasdaq computer network

Security Requirements of cloud

- Three primary cloud security requirements
  - Controlled sharing of information
  - Cloud isolation
  - High Assurance

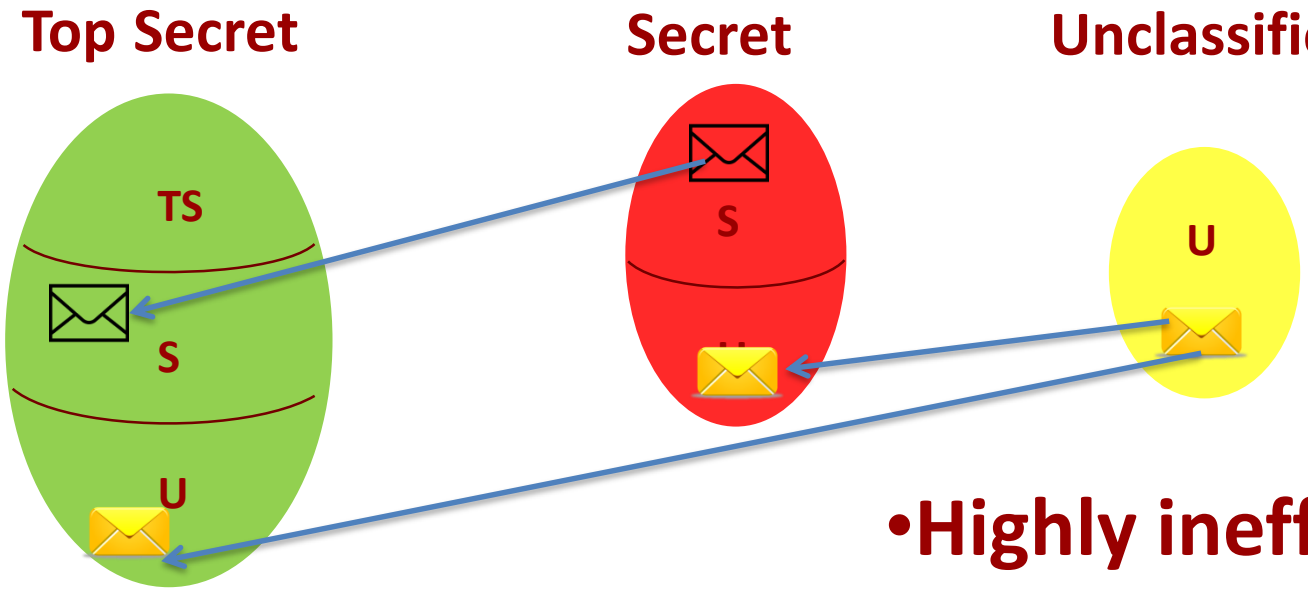
Trap Door Subversion Vulnerability

- Malicious code in platform
  - Software, e.g., operating system, drivers, tools
  - Hardware/firmware, e.g., BIOS in PROM
  - Artifice can be embedded any time during lifecycle
  - Adversary chooses time of activation
- Can be remotely activated/deactivated
  - Unique “key” or trigger known only to attacker
  - Needs no (even unwitting) victim use or cooperation
- Efficacy and Effectiveness Demonstrated
  - Exploitable by malicious applications, e.g., Trojans
  - Long-term, high potential future benefit to adversary
  - Testing not at all a practical way to detect

Alternatives for Controlled Sharing

- Three ways controlled sharing can be facilitated:
- Massive copies of data from all lower levels
  - High assurance one-way flow of information
  - Light diode interface uses physics for high assurance
- File Caching (Local Active Copy)
  - Retain at high level only actually used lower data
  - No way to securely make requests for lower data
  - Security requires manual intervention
- High assurance segmented virtual memory

Massive Copies Approach



- **Highly inefficient!**
- **Does not scale!**

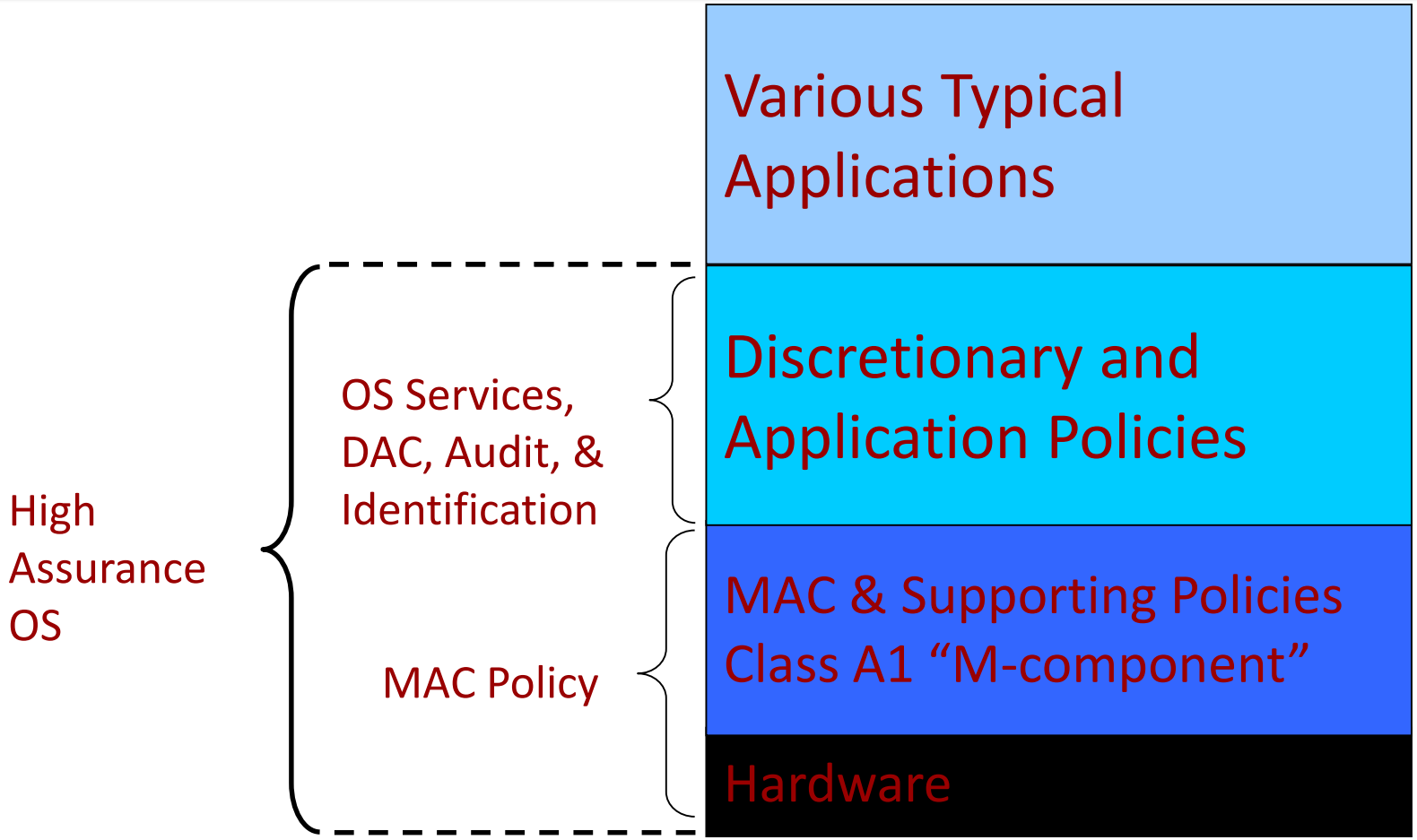
Computer Utility: Predecessor to Cloud

- Computer Utility made security a priority
- Anticipated present day cloud environment
  - Controlled sharing was an integral requirement
  - Incorporated MAC policy
    - Evident from commercial Multics product.
  - Consistent with high assurance
    - Evident from the BLP Multics interpretation.
- Didn't gain widespread acceptance
  - Before Internet enabled today's cloud computing

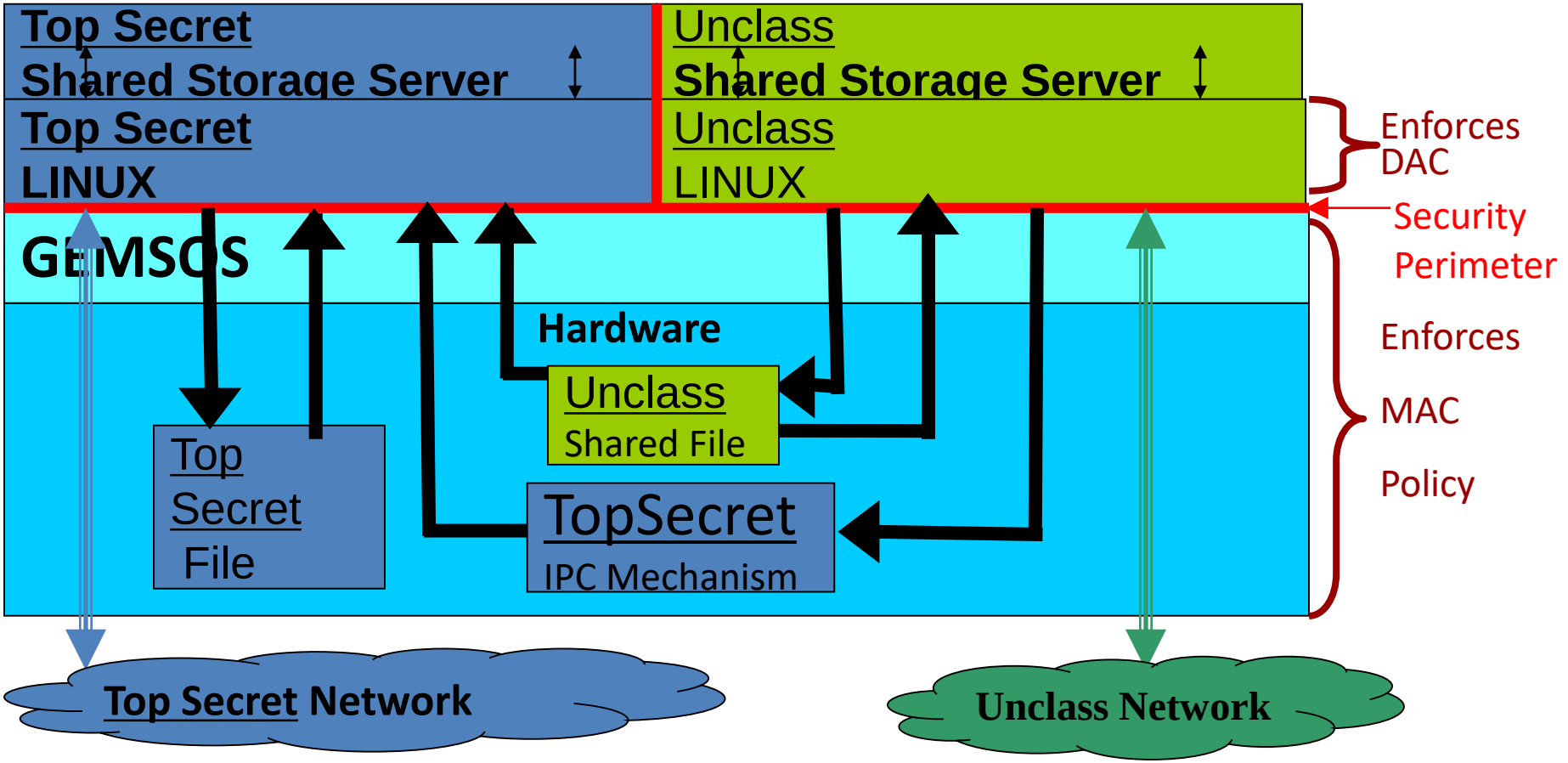
Basis to Consolidate Networks

- Foundation: Trusted Computing Base (TCB)
  - The totality of protection mechanisms
  - within a computer system
  - including hardware, firmware, and software
  - Responsible for enforcing a security policy
- The security perimeter is TCB boundary
  - Software within the TCB is trustworthy
  - Software outside the TCB is untrusted
- Mature technology -- 30 years experience
  - Derived from and used for commercial products
- Key choice is assurance: low to very high

TCB Subsets –"DAC on MAC"



Use Platform for Controlled **Sharing**



Motivation to Address Cloud Security

- Cloud storage flexible, cost-effective
- How to implement in multi-level environment?
  - Duplicate for each level? Loses advantages.
- Tempting target for attackers
  - Can increase privilege
- Want high-assurance, MLS solution
- Want cross-domain sharing (CDS)
  - Share resources are core cloud value proposition
  - Controlled sharing of up to date information
- Solution: MLS cloud network file service
  - Based on high-assurance, evaluated Class A1 TCB

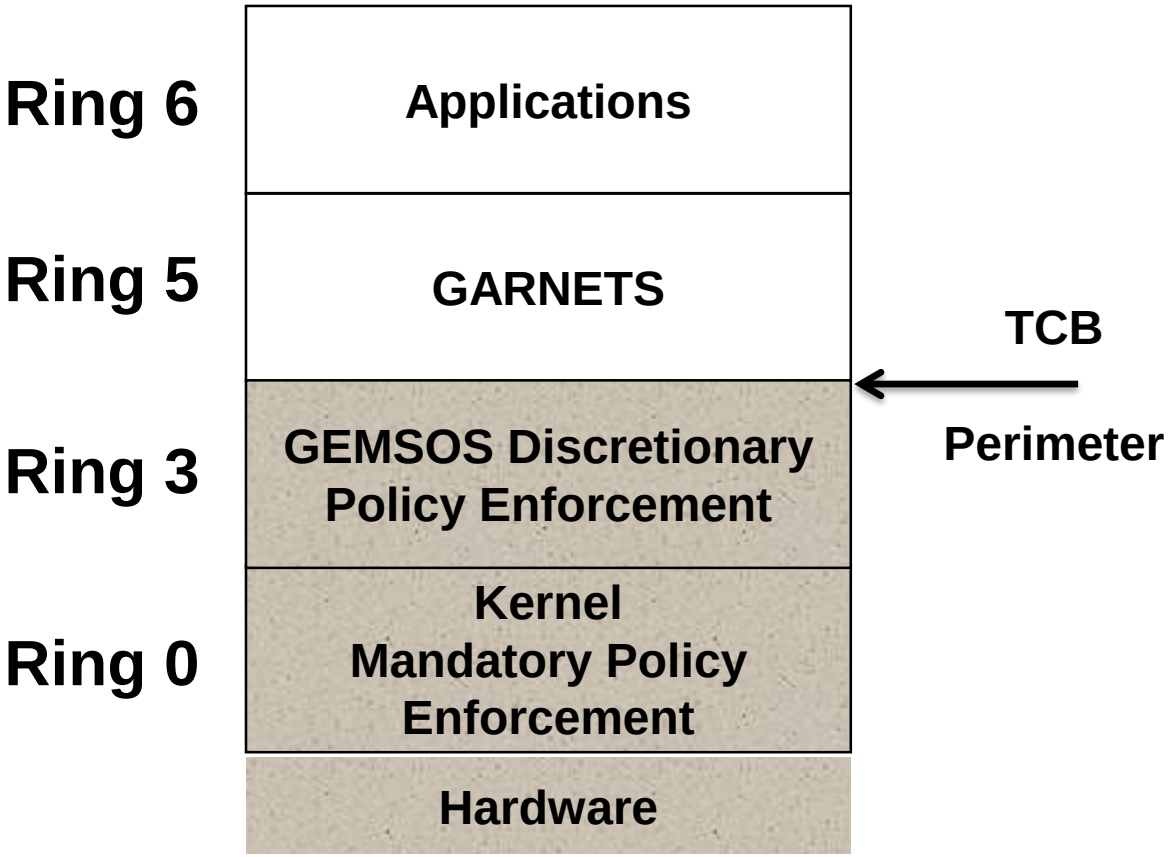
Cloud Storage Security Challenges

- Migrate storage from local security domain
  - Lose protection of “air gap” between domains
  - Dramatically expands opportunity for adversaries
- Common security approaches can’t work
  - Power of insidious software subversion attack tools
  - Exploding “cloud computing” amplifies impact & risk
- Proven reference validation mechanism (RVM)
  - Systematically codified as TCSEC “Class A1”
- Need architecture that leverages trusted RVM
  - Use verifiable (i.e., Class A1) trusted computing base
  - Want high cloud compatibility, e.g., standard NFS

Current Cloud Technology is Vulnerable

- Typically on commodity low-assurance platforms
- NetApp StorageGRID Systems
  - On Linux servers or VMware hypervisors
- OnApp Storage cloud
  - On various hypervisors, including Xen
- Amazon Elastic Compute Cloud
  - On Xen hypervisor to isolate guest virtual machines
- Xen is representative example of low-assurance
  - So-called TCB includes untrustworthy Linux, drivers
  - Largely unconstrained opportunities for subversion
- NSA said system on partition kernel too complex

Review of GARNETS Architecture

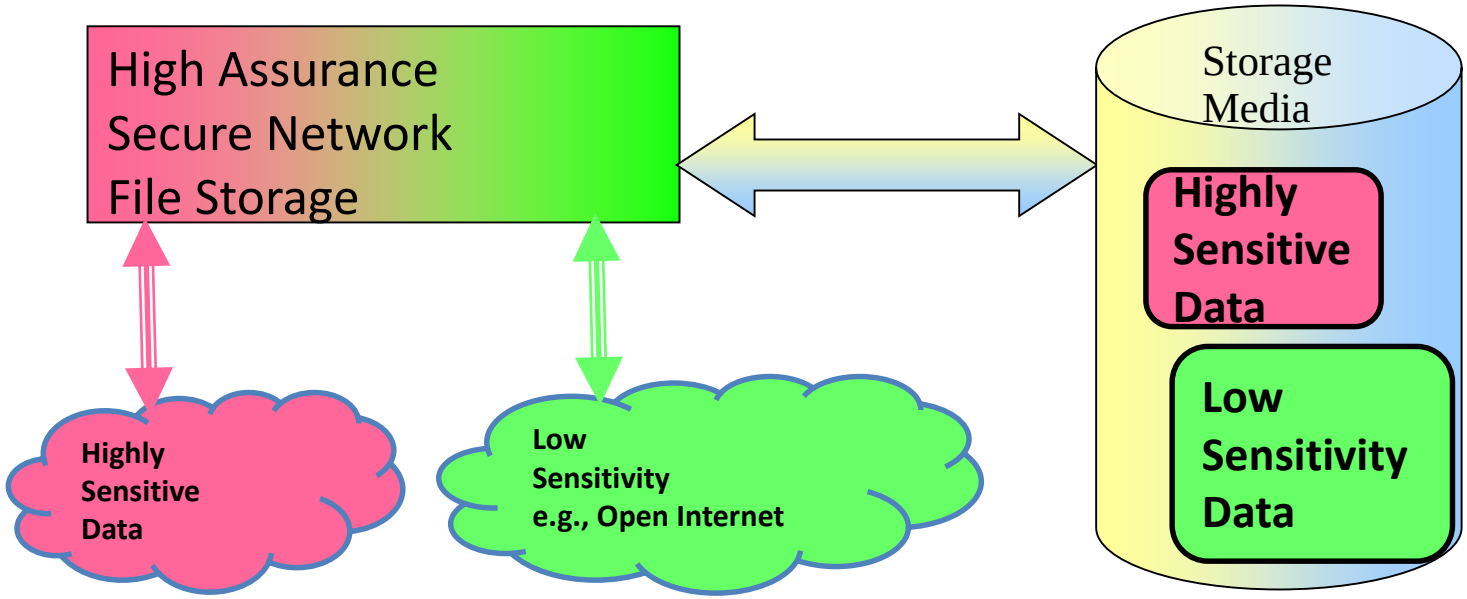


Build on MLS File System Foundation

- GARNETS MLS File is untrusted application
  - In a layer, in a separate ring, above the TCB
  - Creates file management that uses TCB objects
- MLS file system acts like standard file system
  - Spans multiple domains protected by the TCB
  - Creates one logical name space for all domains
- Run GARNETS instances at multiple domains
  - Means no massive copies needed for sharing
  - Can read both directories and files of lower domains
- For cloud storage need to add network interface
  - Multiple network interfaces for multiple domains

Target Secure Cloud Storage

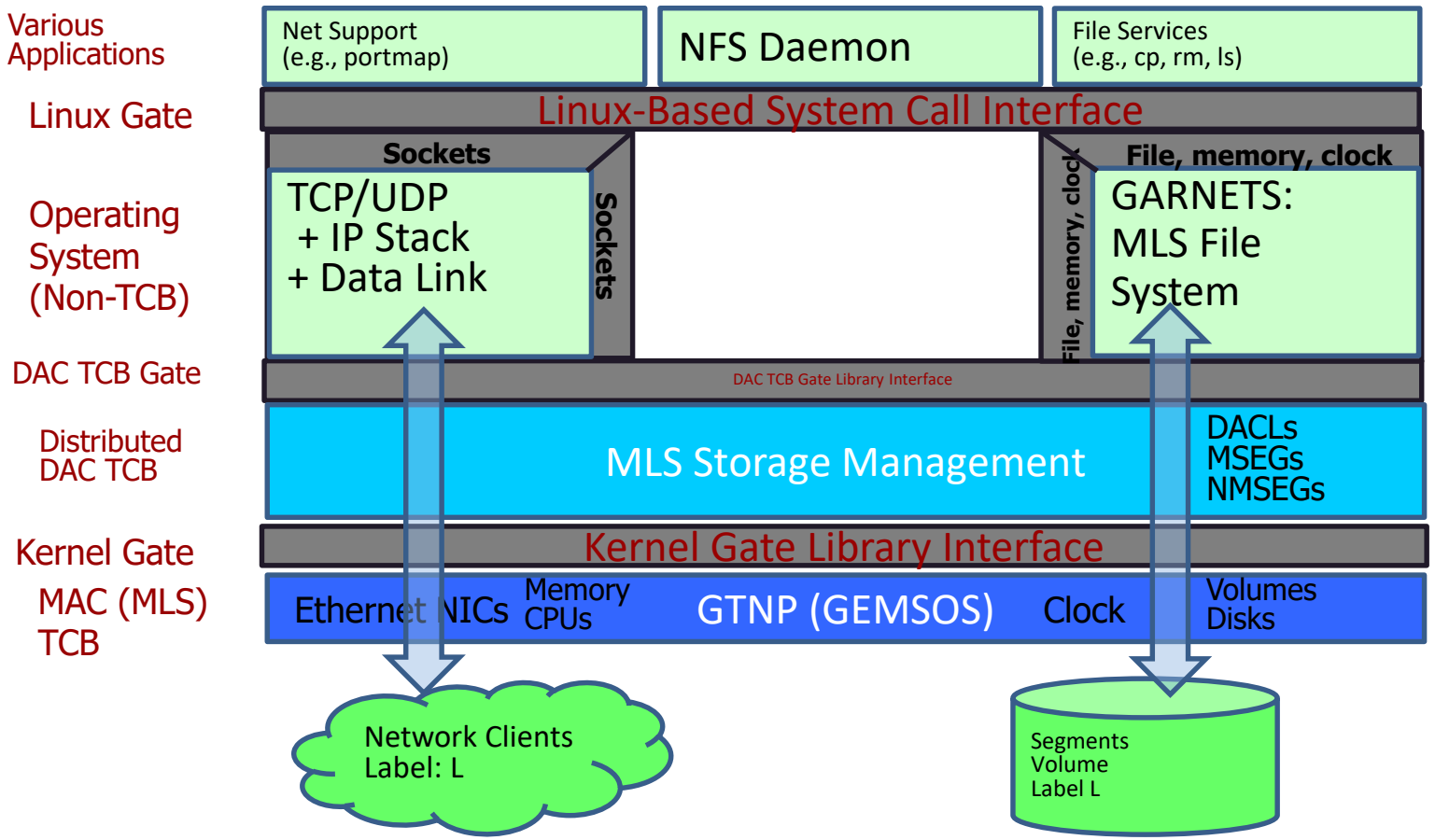
- Operates like standard network file storage
- BUT, verifiable security for
  - MAC separation of security domains



Run NFS on Top of MLS File System

- **System** challenge is a mostly compatible NFS
  - Accept the few intrinsic limitations, e.g., time last read
- Ease of porting NFS wants familiar OS interface
  - Requires creating “compatible OS” [GAS 10.7.2]
  - Demonstration chose POSIX style interface
  - Intended to support Linux source code compatibility
- In contrast, **GARNETS is not compatible**
  - Have to create a Linux system call interface
- Clients access files on server through NSF calls
  - Clients are untrusted
  - Any client using standard NSF protocol can access

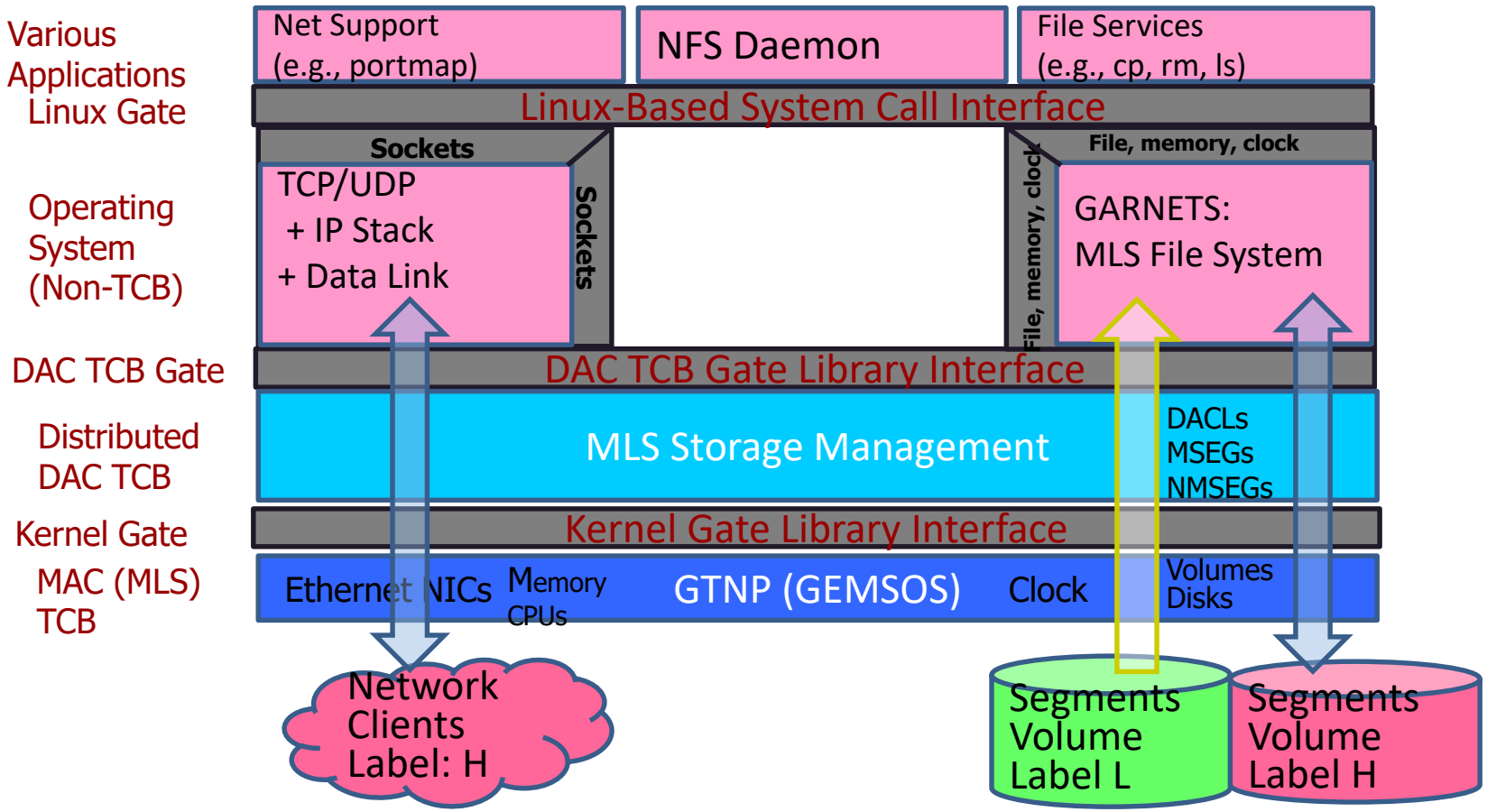
NSF Low Domain S/W Instantiation



For MLS Run Multiple NSF Instances

- Need a separate instance for each level
  - Is the only way NFS can be untrusted software
- Clouds often use virtual machine monitor (VMM)
  - Have noted low-assurance virtual machine problem
- Secure NFS demo leverages GEMSOS VMM
  - FER from NSA describes trusted MLS virtualization
  - Is NOT Type I virtualization, i.e,, cannot run binary
  - Each NFS instance runs in its own virtual machine
- MLS from TCB cannot be bypassed by VMM
  - Can be configured for typical isolation
  - In contrast to most VMMs, has controlled sharing

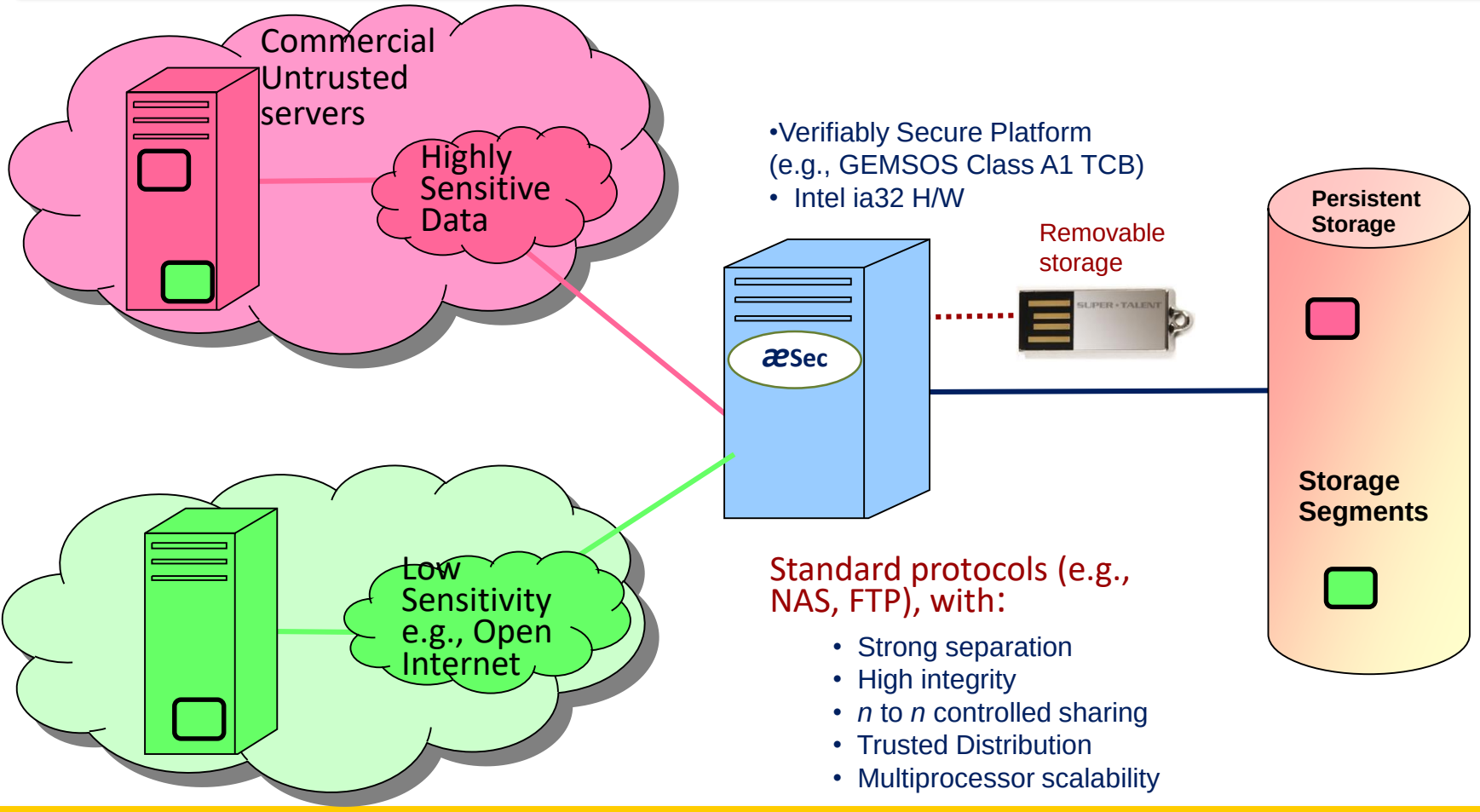
NSF High Domain S/W Instantiation



Can Support Extended Cloud Services

- Usually have metadata server for cloud services
  - Maps object names to actual locations in the cloud
- Single level servers in the network can access
  - May use FTP to transfer files to/from MLS file server
  - Applications NAS protocols can access it files
- BEWARE of sloppy system security engineering
  - Can't initiate information transfers **between** domains
  - TNI provide the network engineering framework
- Separate NIC per domain does not scale well
  - Next look at how to use a single hardware interface
  - Need equivalent of a TCSEC/TNI multilevel device

Security for Untrusted Servers



Cloud Controlled Sharing Summary

- Key is Mandatory Access Control (MAC)
  - Control isolation & sharing between security domains
- Defining properties: global and persistent
  - Control information flow (confidentiality)
    - Prevents malicious information exfiltration
  - Control contaminated modification (integrity)
  - Sound mathematical foundation
- Implement with distinct access class labels
  - Label domain of information with access class
  - User has authorized access classes, i.e., domains
- Supports MAC, viz., multilevel security (MLS)

INF523 SUPLEMENTAL MATERIAL (if time in semester)

Introduction to Crypto Seal Guards Case Study

Professor Clifford Neuman

Suplemental

Crypto Seal Guard Technology History

- Concept: label cryptographically sealed to data
- Conceived ~1980 for AF Korean Air Intelligence
- GEMSOS uses to meet TCSEC “Label Integrity”
  - Gemini Trusted Network Processor (GTNP) (1995)
  - Stored data (disk, tape) in Class A1 Evaluation
- GEMSOS uses for “Trusted Distribution”
  - Authoritative distribution media crypto sealed
  - Only sealed TCB software can be installed and run
- POC applied to packets exchanged by guards
  - Each guard is MLS – both a high and low interface

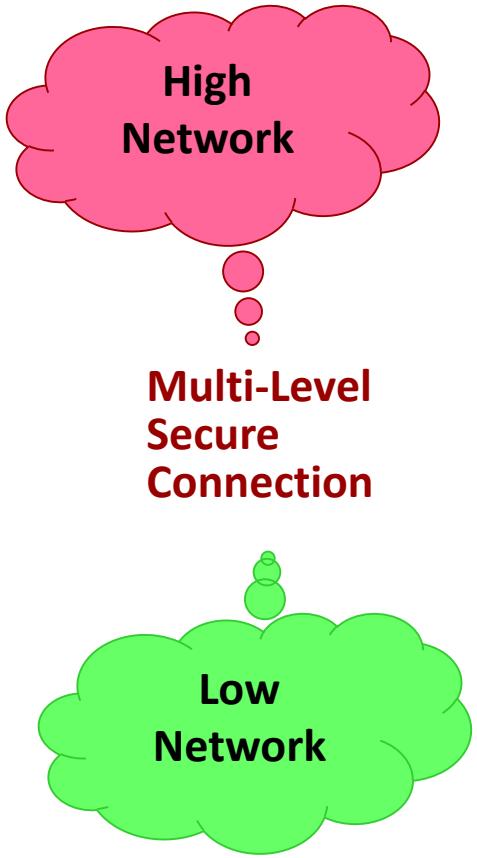
GEMSOS Support for Crypto Seals

- GEMSOS used crypto seals to meet Class A1
  - To meet Class A1 Label Integrity requirements
  - Integral to Trusted Recovery & Trusted Distribution
- GEMSOS publishes security services via APIs:
  - Data Sealing Device (and Cryptographic Services)
  - Key Management
  - Trusted Recovery & Distribution
- GemSeal uses GEMSOS APIs for crypto seals
  - Previously evaluated, stable, public interfaces
  - Minimal new trusted code
    - Generate seal
    - Validate integrity/authenticity of sealed packet & label

Overview of Seals for Shared Networks

- Proof of Concept (POC) demonstration done
  - Crypto seal release guards
  - Preproduction Class A1 MLS platform
- Access low network across system high network
  - Controlled interface protects system high data
  - Vertical data fusion with reduced footprint
- Benefits of crypto seal release guards
  - Swift implementation for MLS systems
  - Available core enabling technology for MLS
  - Rapid path to certification and accreditation (C&A)
  - Supports entire range of security domains
  - Mature deployed NSA Class A1 TCB and RAMP plan

Constraints to Access Lower Networks



- Any low connection = Multi-Level
  - Must be Multi-Level Secure (MLS)
  - Low/Medium assurance ineffective
    - Doesn't protect against subversion
    - Vulnerabilities unknown (unknowable)
- Isolation obstructs missions
  - Vertical data fusion
  - Tactical situational awareness
  - Timely access to open source data
  - Efficient utilization of resources

GemSeal POC Uses MLS Technology

- Class A1 TCB - GEMSOS™ security kernel
- Class A1 Ratings Maintenance Plan (RAMP)
- MLS aware crypto seal release guard
  - Gemini call it the GemSeal™ concept
- Technology Benefits
  - Minimize new trusted code development
  - Extensible to gamut of MLS capable systems
- High assurance resists subversion
  - Verifies absence of malicious code
  - Effective application of science
  - Key enabler for demanding accreditation, e. g., PL-5

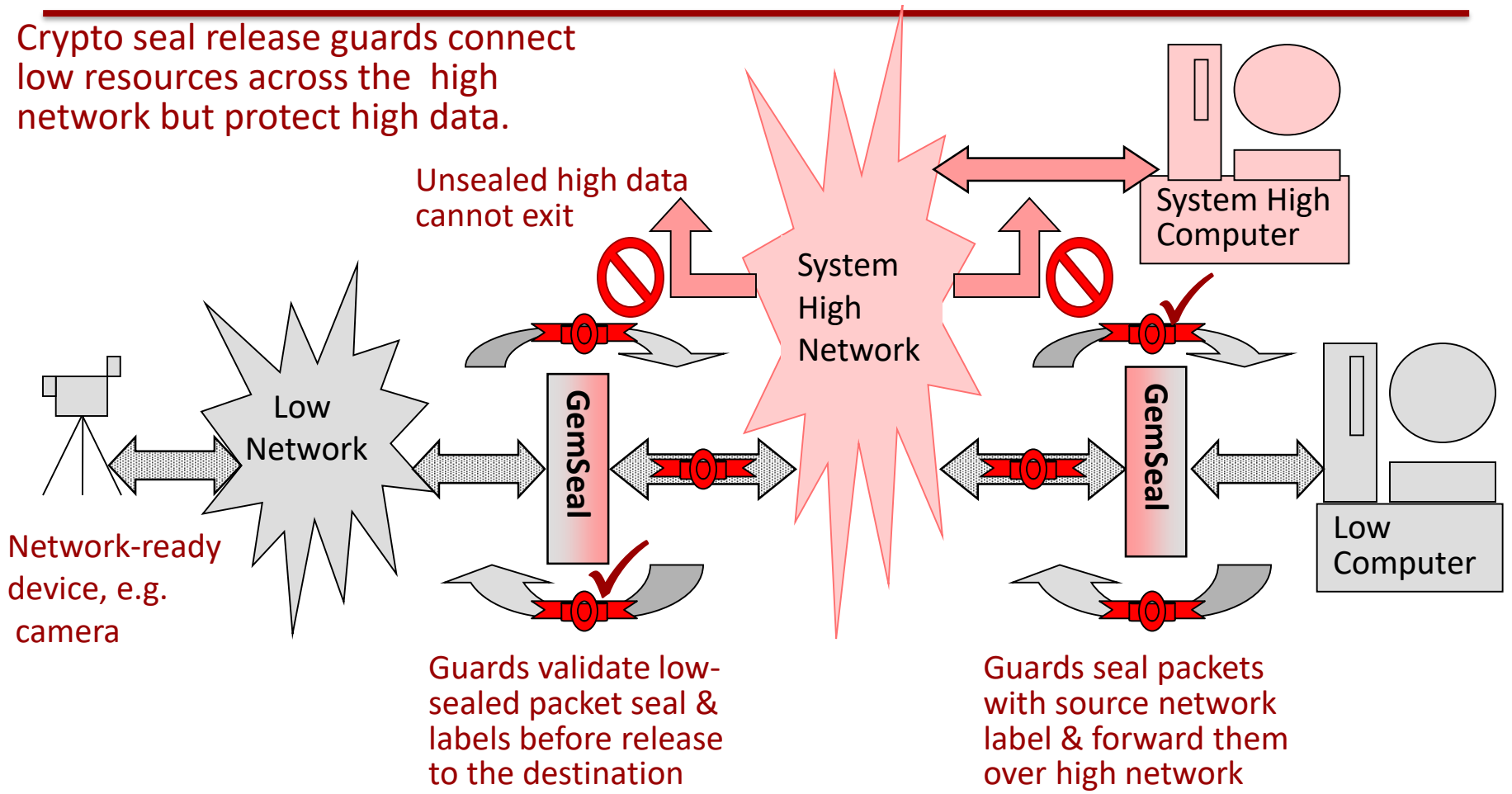
How Guard Seals a Packet

- Packet switched network design, e.g. Internet
- Concept involves multiple guards
  - POC has one or more “workstation” guards
  - POC has one or more “sensor” guards
  - Connected via a common system-high network
- Each guard has both high and low interfaces
- Sealing packets – forwarding from low to high
  - Bind source interface (low) label to each packet
  - Generate cryptographic seal of packet data + label
  - “Low-Sealed” packets include packet data + seal
  - “Low-Sealed” packets via high network interface

How Guard Releases a Packet

- Releasing packets – delivering from high to low
  - Release ONLY packets with seal-validated labels
  - Seal and label are removed before being released
- Only released to interfaces matching labels
  - Allows low data to traverse & exit high network
  - Concept supports multiple release guards
- Assures integrity of BOTH data AND label
  - Packet data is not altered
  - Source sensitivity label is authentic for this packet

AF Crypto Seal POC Demonstration



Summary of AF POC Demonstration

- Sensor (video) stream + command and control
  - Low sensor to low workstation connectivity
    - Uses existing high network infrastructure
  - Delivers access to low devices
    - For users lacking low network infrastructure
    - From controlled interface
- High network data is protected and unchanged
  - Guard validates low-sealed packets before release
  - Unsealed high packets cannot exit via guard

Summary of POC Configuration

- Two untrusted workstations with browsers
  - One ("Low") connected to "workstation guard"
  - One ("High") connected to high network
- One web server
  - Connected to low-side of the "sensor guard"
- A "high" Ethernet LAN
  - Connected to high-side of both guards
  - Also connected to second system high workstation
- The demonstration shows that
  - "Low" workstation can browse the "Low" web server
  - "High" workstation has no access to "Low" web server

Prior Evaluation Aids Accreditors

- Simplify job with reusable accreditation results
  - Certify or assess the platform once
  - Focus on system-specific additions & configurations
- GTNP provides evaluable TCB platform
  - Previously evaluated Class A1 for TNI M-Component
  - Class A1 RAMP in place and already proved useful
- Outside of the GTNP trusted computing base
  - Most of the application software will be untrusted
  - Only cryptographic seal operations need be trusted
    - Generate seals & release packets with validated seals
- Customer's certification and accreditation needs
  - The verifiably secure MLS TCB and

POC to Deployable System Summary

- Don't have to evaluate platform first
  - RAMP is already proven
  - No formal specification changes anticipated
- First: Evaluate and accredit the parts separately
  - Platform (very stable, accredit new hardware ports)
  - Crypto Seal implementation (as a trusted application)
  - Guard applications themselves evaluated separately
    - Supporting policies - audit, DAC, etc.
    - Untrusted application pieces, including network stack
    - Each protected by security kernel
- Last: refresh platform evaluation + accreditation
  - Because already successfully evaluated & accredited

Introduction to RECON Guard Security

- Review a classic and seminal paper
  - Cite: J. P. Anderson, "On the Feasibility of Connecting RECON to an External Network," Technical Report, J. P. Anderson Co., March 1981
  - Often cited for both databases and communications
- RECON is on-line interactive data base
  - Citations for both raw and finished intelligence reports
  - Also overnight batch and canned query capability
  - User may specify which file(s) to search
- Sponsor's security concerns are twofold
  - Subject to penetration from external network
  - Spillage of sensitive information from internal failure

Data Security Protection

- The data base contains two kinds of records
  - Those which can be widely distributed
  - Those whose distribution is restricted
    - Compartmented
    - Proprietary
    - Originator-controlled
- Operative aspects of the security problem
  - Commodity mainframe operating system
  - Must be prudently assumed that trapdoors exist
    - In some or much of application or operating systems
    - May be activated from externally connected users

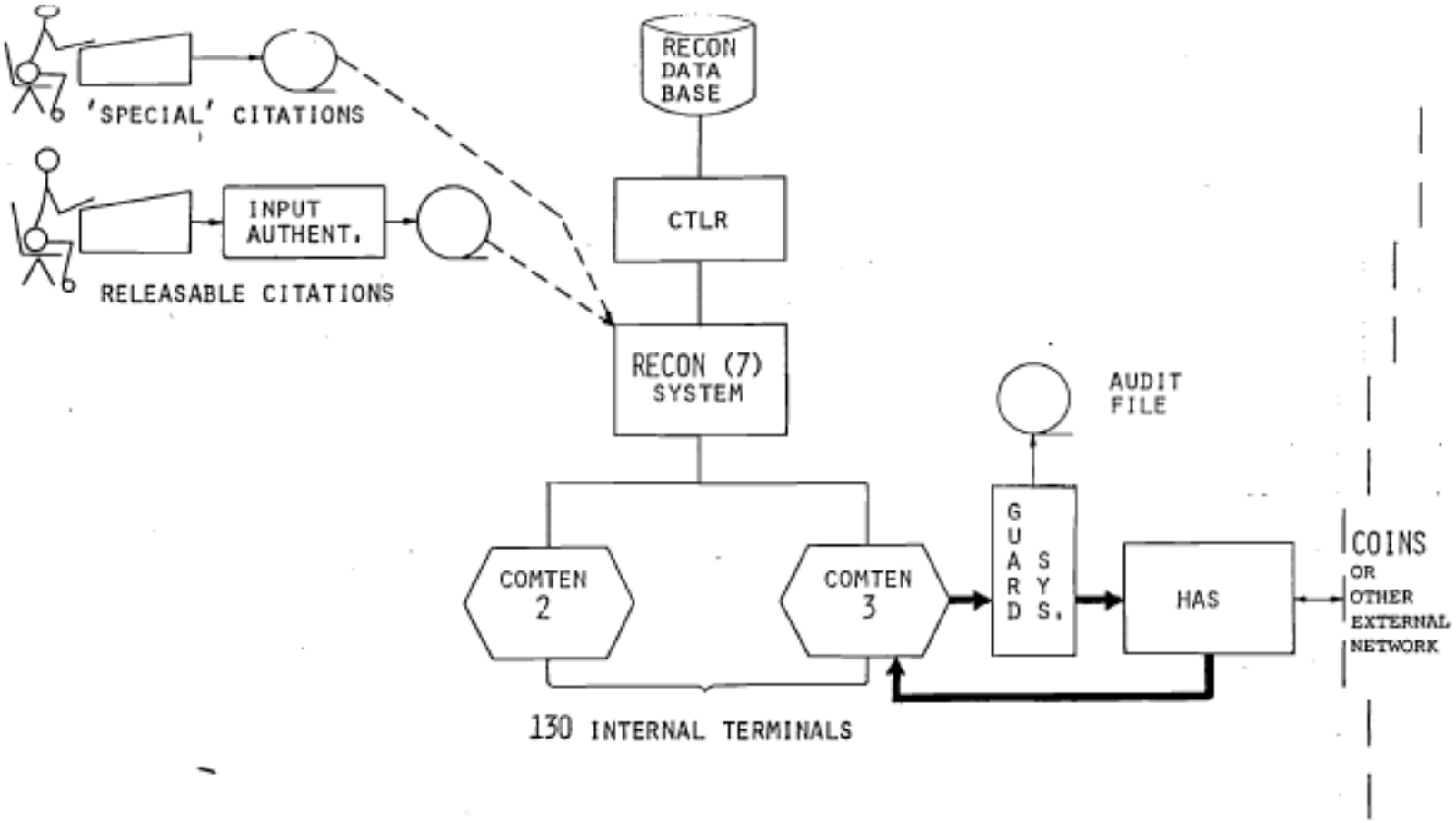
Previously Considered Approaches

- Put two kinds of records in separate systems
  - Make entries not deemed "special" accessible
  - Protected the sponsor's assets from penetration
  - Rejected because of the cost of duplicate facilities
- Multilevel secure operating system
  - In principle, would go far to defeat direct attacks
  - Could defeat placing trapdoors and Trojan Horses
  - Produce totally incompatible (with anything!) systems
  - Very expensive
- Filters added to RECON software to limit access
  - Nothing to control internal or external penetration

Guard Authenticate Releasability

- Is akin to the problem of "sanitizing" SCI
  - For release to activities without proper clearances
- Permit arbitrary queries by all users
  - Route query result of uncleared users to sanitizer
  - Sanitization officer would manually examine output
- Sanitization officer approach works in principle
  - Not practical solution because of excessive delays
  - Delays cascade to produce large response times
- Adapted as proposal to solve RECON problem
  - Adopt the idea of “sanitization” in a GUARD station
  - Automate the identification of releasable citations

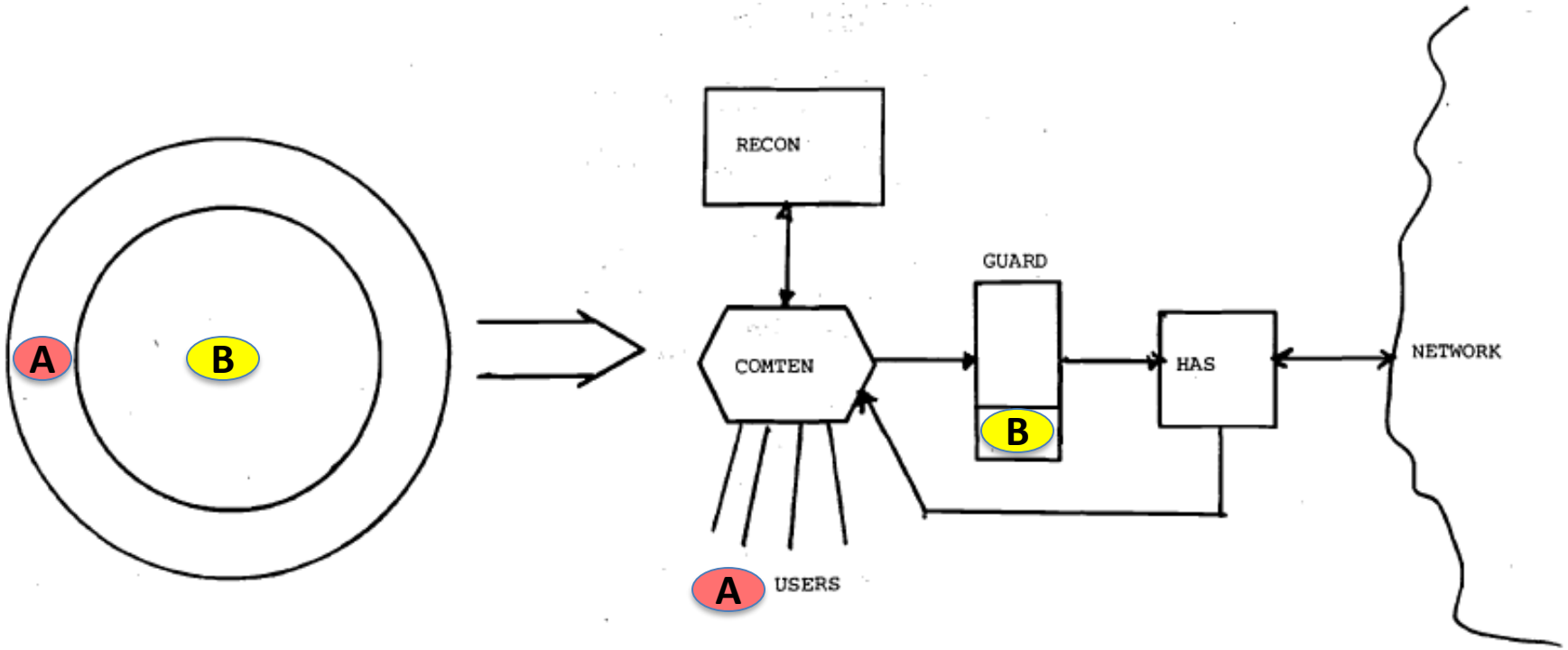
RECON Guard Technical Approach



RECON Guard Concept of Operation

- Consider all citations in one of two cases
  - Releasable even if not approved for "special" citations
  - Releasable only to approved individuals
- Each RECON entry designated by **originator**
  - Whether (or not) it is releasable to external users
- Create cryptographic checksum for releasable
  - Computed as the data enters the system
  - A function of the entire record
  - Computed by a special authentication device
  - Checksum is appended to the record and stays with it

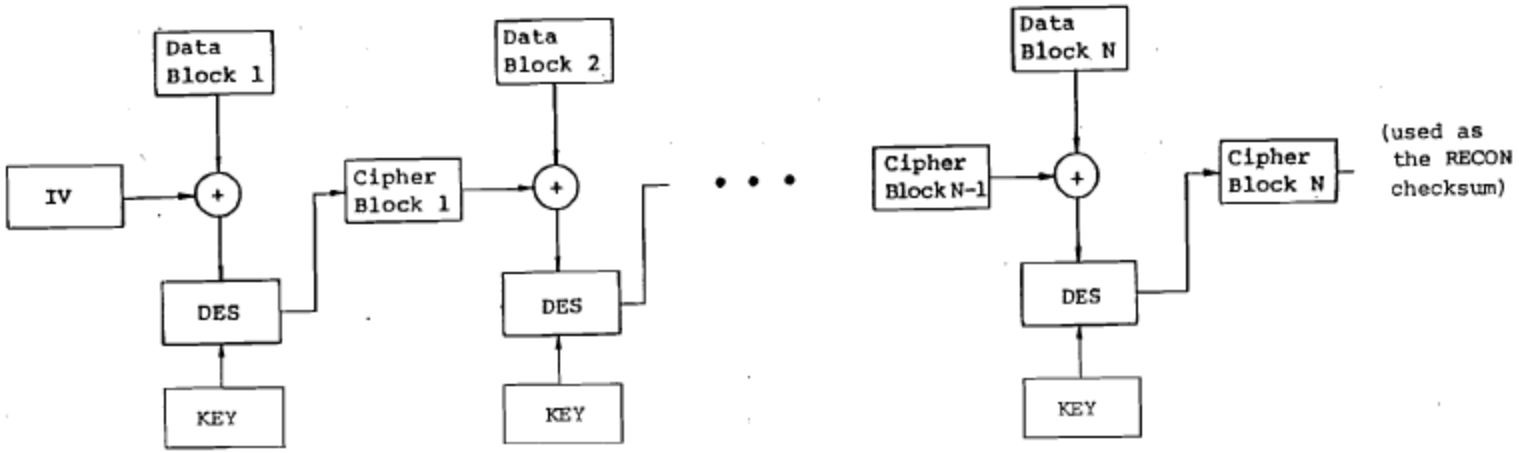
Representation of Basic Capability



Cryptographic Checksums Properties

- Principle need is assuring checksum not forged
  - Good modern crypto algorithm
  - Perform checksum functions outside RECON hosts
    - Separate entities to create and do Guard functions
- Secret key is known only to checksum devices
  - Key is never available within RECON system
  - Hardwired on board with crypto processor
  - Only method to forge is random guess (brute force)
- Key used for block-chained encipherment
  - Excellent error or tampering detection
  - Initial variable (IV) is used as half of the “secret”
  - A security “kernel” in devices control their operation

Process to Create Crypto Checksum



- ⊕ Exclusive OR
- The secret key(s) are the Initial Variable (IV) and the KEY.

BLOCK CHAINING

Security Properties of Guard

- No spill from RECON failure or compromise
- No manipulation of RECON will cause release
- Will “fail safe” if checksum detached from data
- Not protecting against manipulation of data
- Not preventing denial of service
- Guard system itself defends against its failure
  - Advanced design techniques, e.g., formal specs
  - Programs placed in read-only memory
  - Permits RECON to test guard message w/ loop back