CIS 551 / TCOM 401

Computer and Network Security

Spring 2006

Lecture 6

Announcements

- Reminder:
  - Project 1 is due TODAY
  - Mail your .tar file to Karl by midnight tonight.

- Some of today's slides are adapted from slides by John Mitchell

Recap from last time

- We've been studying Acess Control Mechanisms
  - Access control lists
  - Capabilities
  - Unix/Windows OS access control
  - Stack inspection

- Today:
  - Discretionary access control (DAC)
  - Mandatory access control (MAC)
  - Information-flow security

Access Control

- *Discretionary*: The individual user may, at his own discretion, determine who is authorized to access the objects he creates.

- *Mandatory*: The creator of an object does not necessarily have the ability to determine who has authorized access to it.
  - Typically policy is governed by some central authority
  - The policy on an object in the system depends on what object/information was used to create the object.
  - Examples?

Multilevel Security

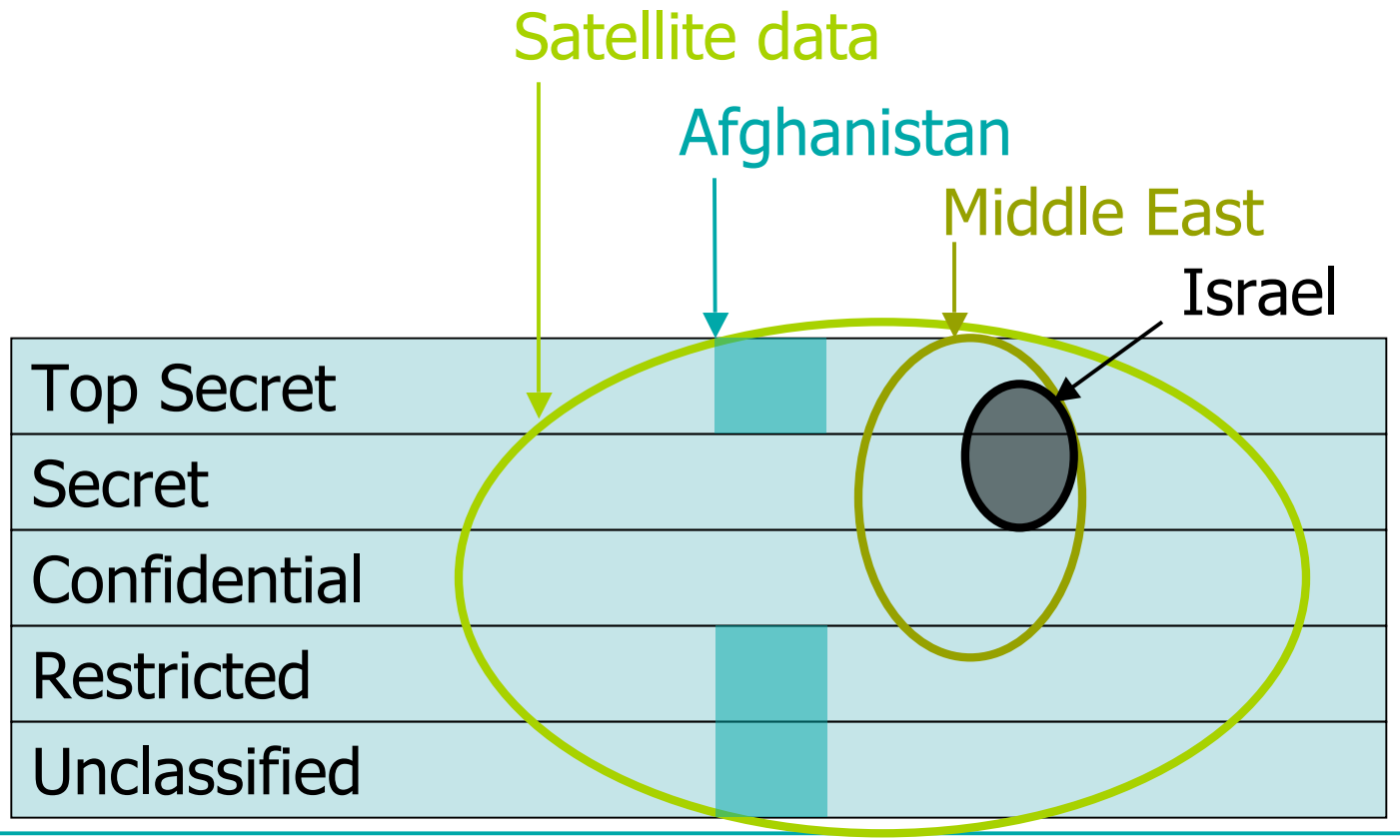
- Multiple levels of confidentiality ratings
- Military security policy
    - Classification involves sensitivity levels, compartments
    - Do not let classified information leak to unclassified files
- Group individuals and resources
  – Use some form of hierarchy to organize policy

- Trivial example: Public ≤ Secret

- *Information flow*
  – Regulate how information is used throughout entire system
  – A document generated from both Public and Secret information must be rated Secret.
  – Intuition: "Secret" information should not flow to "Public" locations.

Military security policy

- Sensitivity levels
- Compartments



Military security policy

- Classification of personnel and data
  - Class D = ⟨rank, compartment⟩
- Dominance relation
  - $D_1 \leq D_2$ iff $rank_1 \leq rank_2$ and $compartment_1 \subseteq compartment_2$
  - Example: ⟨Restricted, Israel⟩ ≤ ⟨Secret, Middle East⟩
- Applies to
  - Subjects – users or processes: C(S) = "clearance of S"
  - Objects – documents or resources: C(O) = "classification of O"

Bell-LaPadula Confidentiality Model

- "No read up, no write down."
  - Subjects are assigned clearance levels drawn from the lattice of security labels.

    C(S) = "clearance of the subject S"
  - A principal may read objects with lower (or equal) security label.
    - Read: $C(O) \leq C(S)$
  - A principal may write objects with higher (or equal) security label.
    - Write: $C(S) \leq C(O)$

- Example:
  A user with Secret clearance can:
  - Read objects with label Public and Secret
  - Write/create objects with label Secret

Multilevel Security Policies

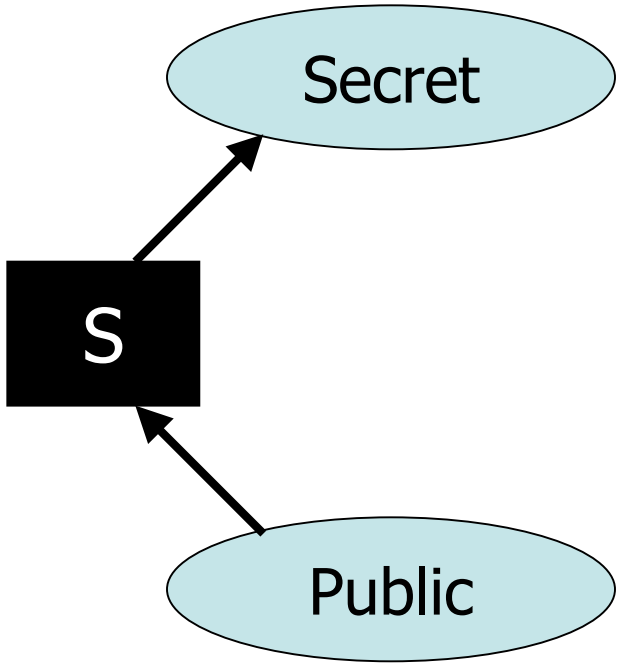
- In general, security levels form a "join semi-lattice"
  - There is an ordering ≤ on security levels
  - For any pair of labels L1 and L2 there is an "join" operation:

    L1 ⊕ L2 is a label in the lattice such that:
    (1) L1 ≤ L1 ⊕ L2 and L2 ≤ L1 ⊕ L2 "upper bound"
    (2) If L1 ≤ L3 and L2 ≤ L3 then L1 ⊕ L2 ≤ L3 "least bound"

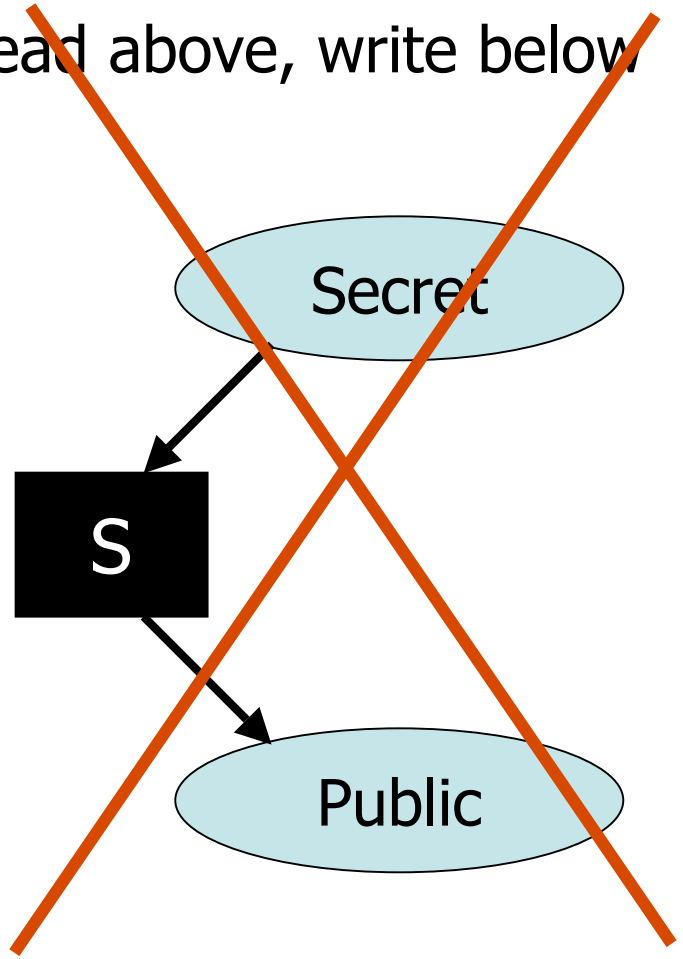
- For example: Public ⊕ Secret = Secret
- Labeling rules:
  - Classification is a function C : Object → Lattice
  - If some object O is "created from" objects $O_1,\ldots,O_n$ then $C(O) = C(O_1) \oplus \ldots \oplus C(O_n)$

Picture: Confidentiality

Read below, write above

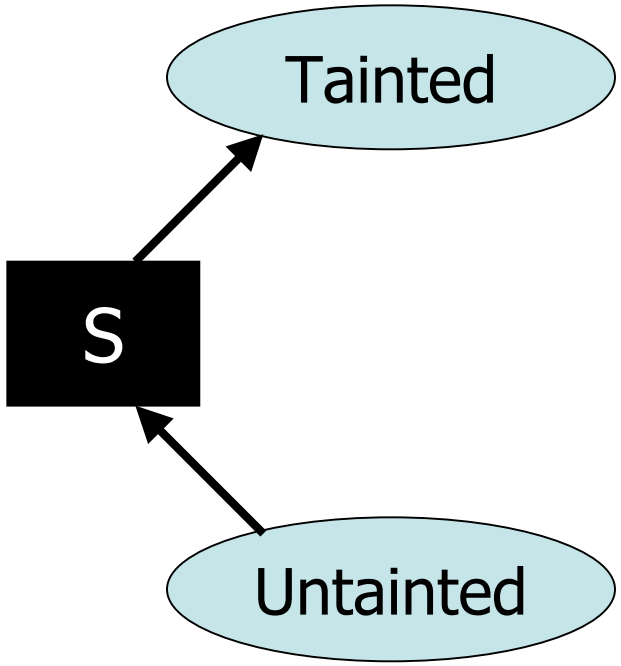


Read above, write below

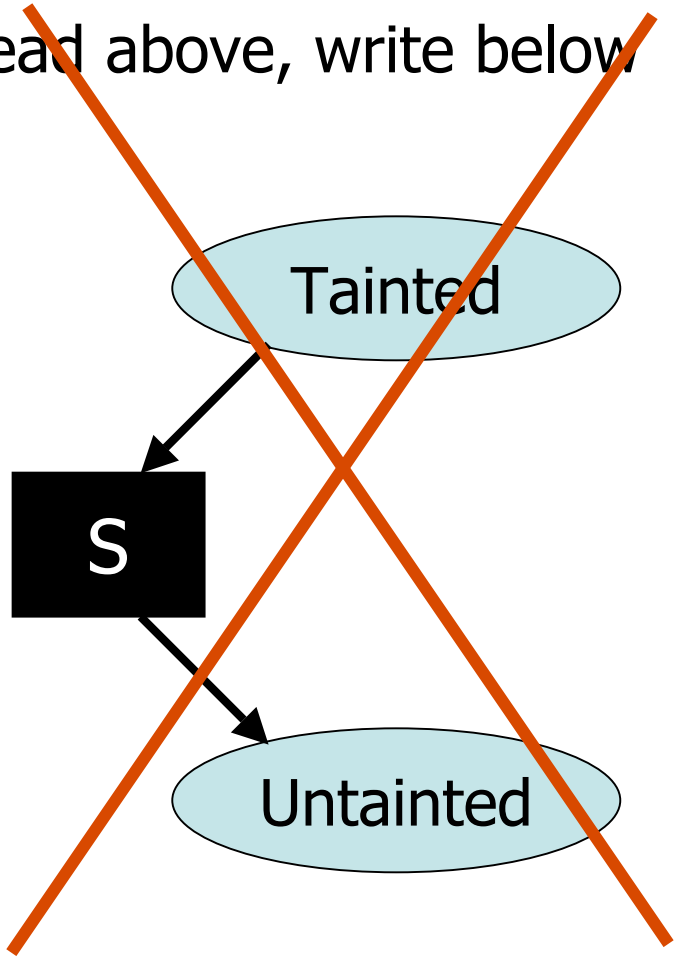


Picture: Integrity

Read below, write above



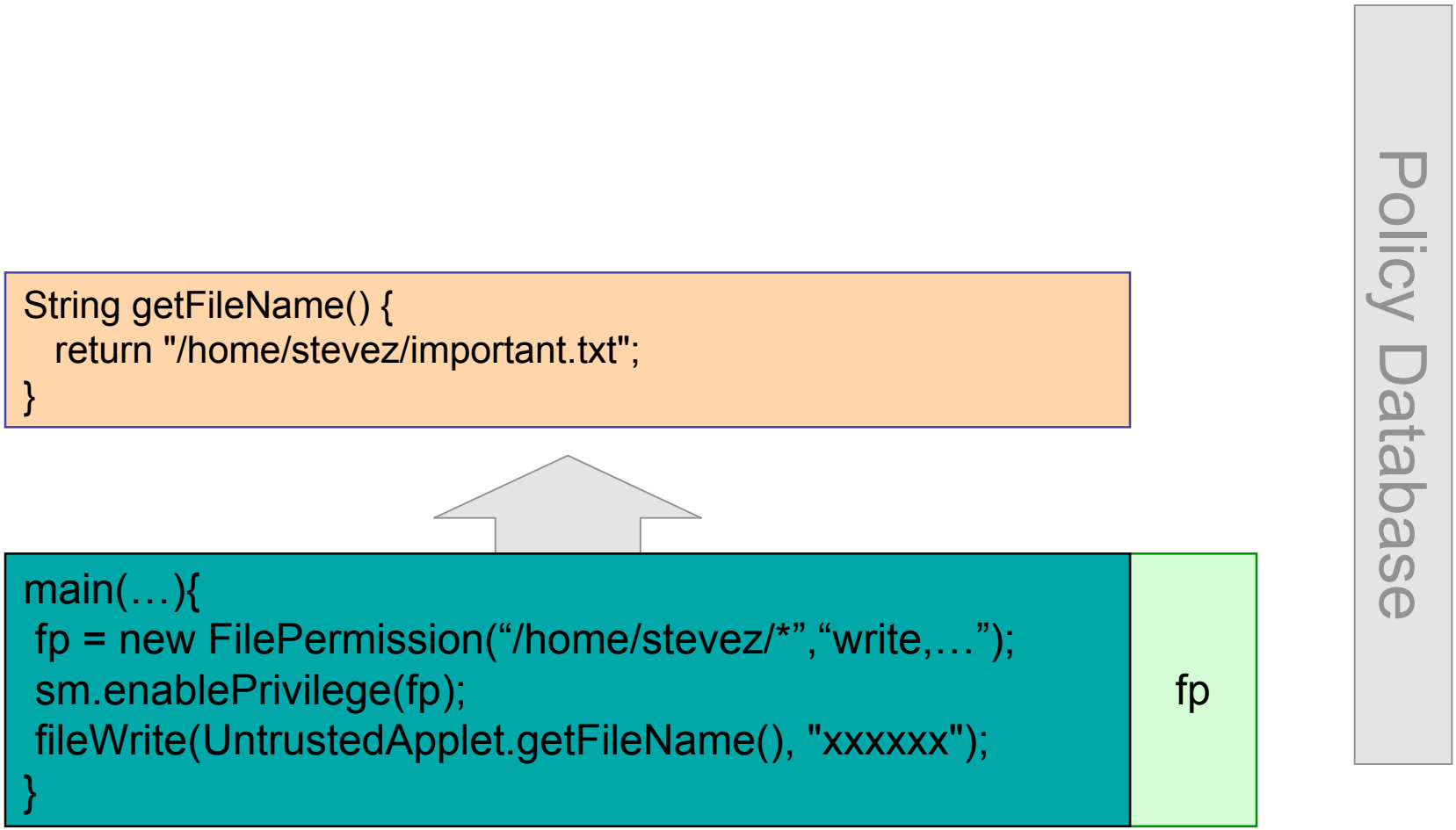
Read above, write below



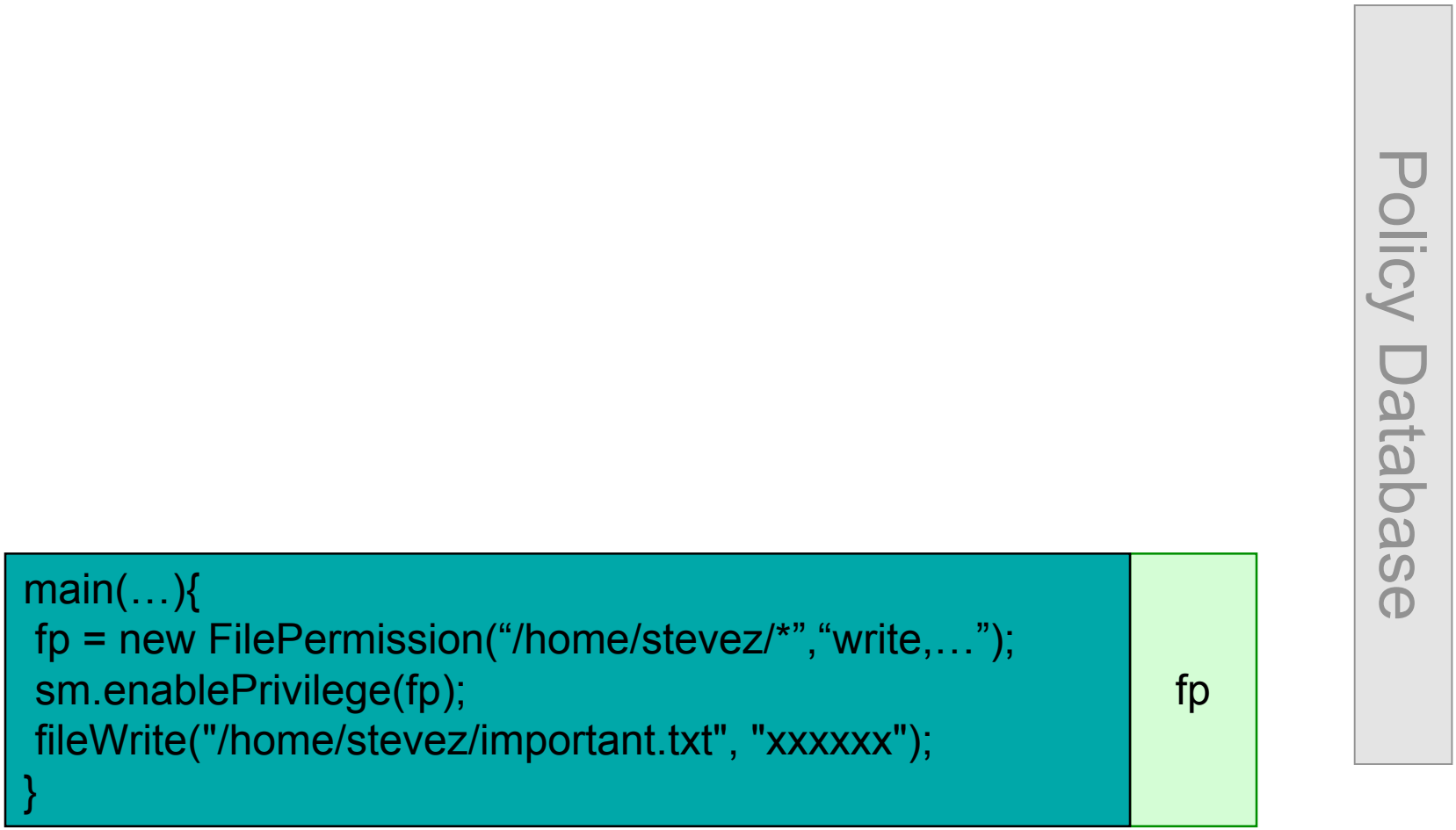
Problem with Stack Inspection

Problem with Stack Inspection

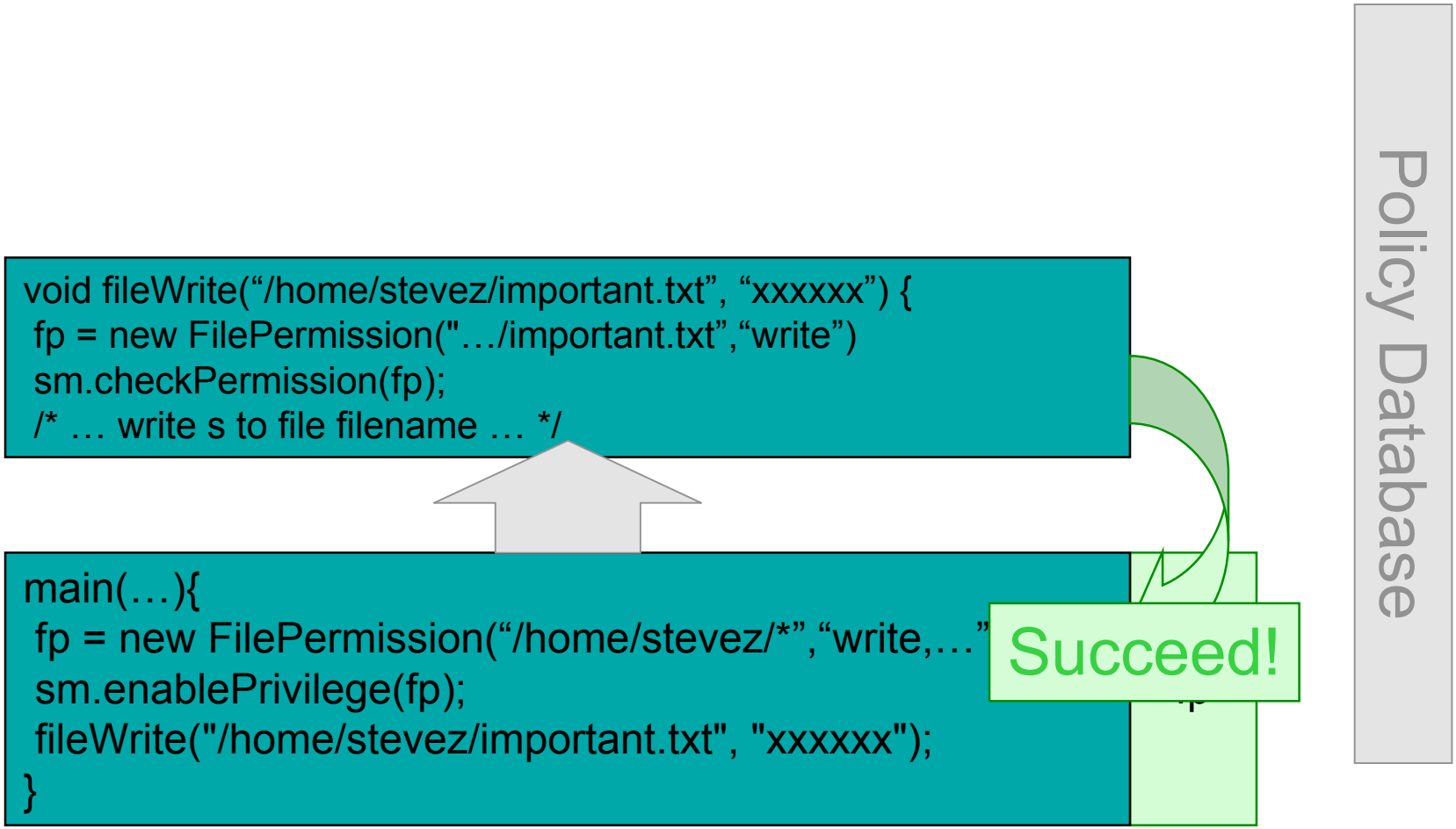
Problem with Stack Inspection



Problem with Stack Inspection



Problem with Stack Inspection



Implementing Multilevel Security

- Dynamic:
  - Tag all values in memory with their security level
  - Operations propagate security levels
  - Must be sure that tags can't be modified
  - Expensive, and approximate

- Classic result: Information-flow policies cannot be enforced purely by a reference monitor!
  - Problem arises from implicit flows

- Static:
  - Program analysis
  - May be more precise
  - May have less overhead

Information Flows through Software

Explicit Flows:

```
int{Secret} X = f();
int{Public} Y = 0;

Y = X;
```

Implicit Flows:

```
int{Secret} X = f();
int{Public} Y = 0;
int{Public} Z = 0;

if (X > 0) then {
  Y = 1;
} else {
  Z = 1;
}
```

Perl's Solution (for Integrity)

- The problem: need to track the source of data
- Examples: Format string, SQL injection, etc.

```
$arg = shift;
system ("echo $arg");
```

- Give this program the argument `"; rm *"`
- Perl offers a *taint checking* mode
  - Tracks the source of data (trusted vs. tainted)
  - Ensure that tainted data is not used in system calls
  - Tainted data can be converted to trusted data by pattern matching
  - Doesn't check implicit flows

SELinux

- Security-enhanced Linux system (NSA)
  - Enforce separation of information based on confidentiality and integrity requirements
  - Mandatory access control incorporated into the major subsystems of the kernel
    - Limit tampering and bypassing of application security mechanisms
    - Confine damage caused by malicious applications

http://www.nsa.gov/selinux/

SELinux Security Policy Abstractions

- Security-Encanced Linux
  - Built by NSA
- Type enforcement
  - Each process has an associated domain
  - Each object has an associated type (label)
  - Configuration files specify
    - How domains are allowed to access types
    - Allowable interactions and transitions between domains
- Role-based access control
  - Each process has an associated role
    - Separate system and user processes
  - configuration files specify
    - Set of domains that may be entered by each role

Two Other MAC Policies

- "Chinese Wall" policy: [Brewer & Nash '89]
  - Object labels are classified into "conflict classes"
  - If subject accesses one object with label L1 in a conflict class, all access to objects labeled with other labels in the conflict class are denied.
  - Policy changes dynamically

- "Separation of Duties":
  - Division of responsibilities among subjects
  - Example: Bank auditor cannot issue checks.