TEL AVIV UNIVERSITY

Information Security – Theory vs. Reality

0368-4474-01, Winter 2011

Lecture 7: Information flow control

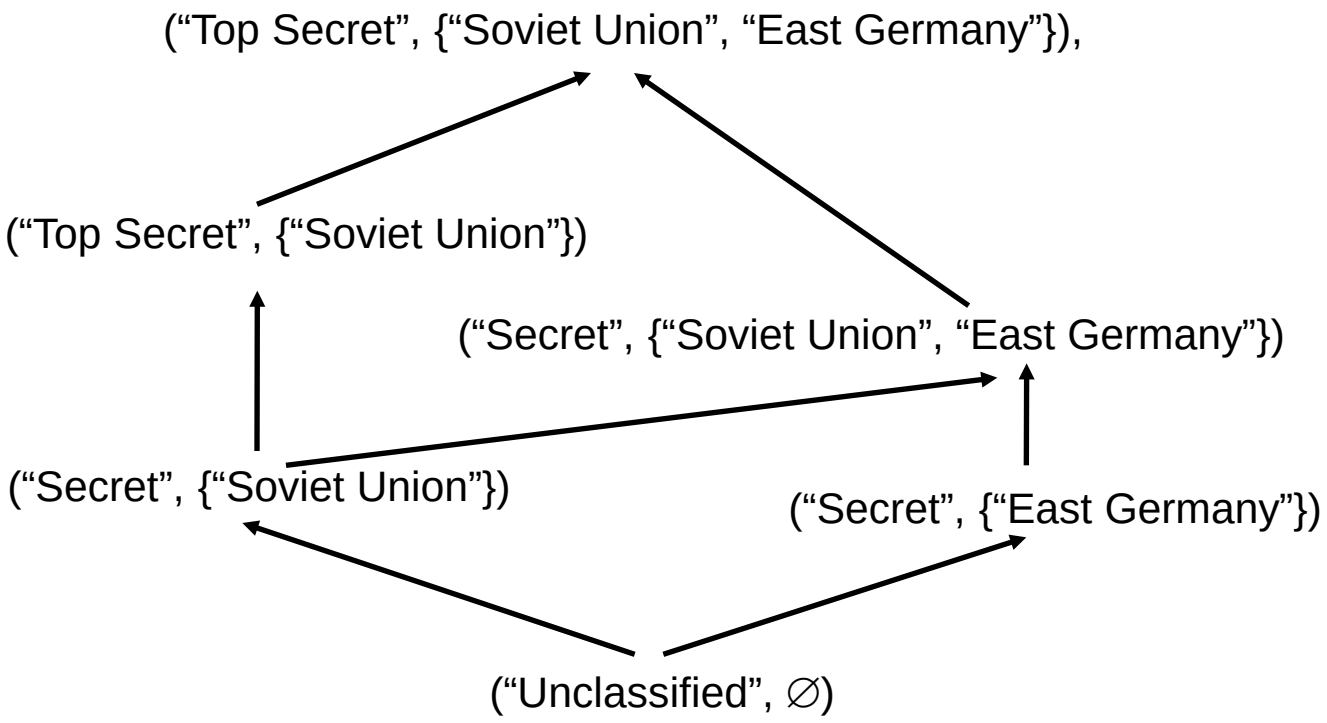
Eran Tromer

Slides credit:
Max Krohn, MIT
Ian Goldberg and Urs Hengartner, University of Waterloo

Information Flow Control

- An information flow policy describes authorized paths along which information can flow
- For example, Bell-La Padula describes a <u>lattice-based</u> information flow policy

Example Lattice



Lattices

- Dominance relationship ≥ defined in military security model is transitive and antisymmetric
- Therefore, it defines a lattice
- For two levels a and b, neither a ≥ b nor b ≥ a might hold
- However, for every a and b, there is a lowest upper bound u for which u ≥ a and u ≥ b, and a greatest lower bound l for which a ≥ l and b ≥ l
- There are also two elements U and L that dominate/are dominated by all levels
  - In example,
    U = (“Top Secret”, {“Soviet Union”, “East Germany”})
    L = (“Unclassified”, ∅)

Information Flow Control

- Input and output variables of program each have a (lattice-based) security classification S() associated with them
- For each program statement, compiler verifies whether information flow is secure
- For example, `x = y + z` is secure only if $S(x) \geq lub(S(y), S(z))$, where lub() is lowest upper bound
- Program is secure if each of its statements is secure

Implicit information flow

Conditional branches carry information about the condition:

```
secret bool a;
public bool b;
…
b = a;
if (a) {
    b = 0;
} else {
    b = 1;
}
```

Possible solution: assign label to program counter and include it in every operation. (Too conservative!)

Issues

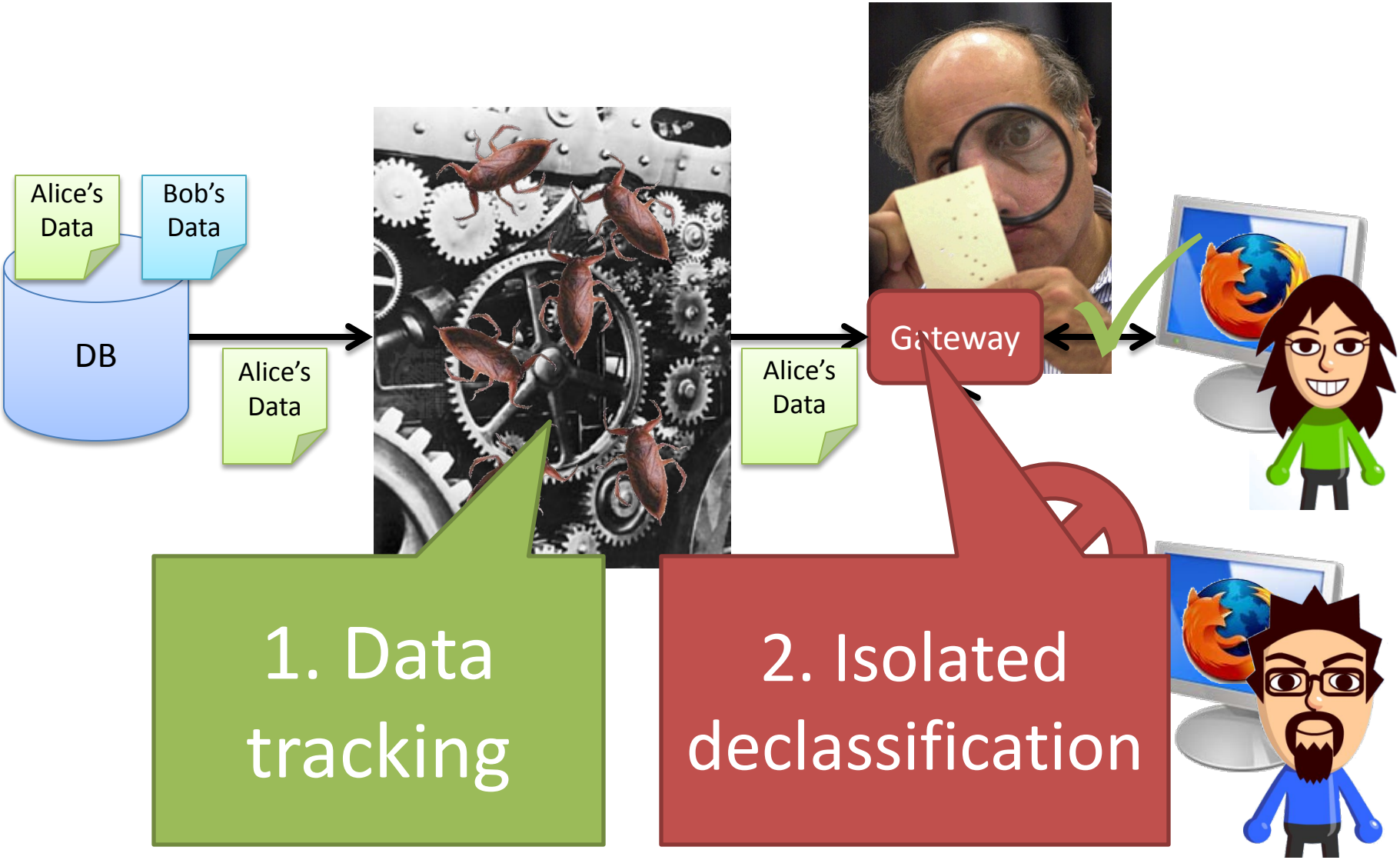
- Label creep
  - Example: false flows
  - Example: branches

- Declassification
  - example: encryption

Implementation approaches

- Language-based IFC (JIF & others)
  - Compile-time tracking for Java, Haskell
  - Static analysis
  - Provable security
  - Label creep (false positives)
- Dynamic analysis
  - Language / bytecode / machine code
  - Tainting
  - False positives, false negatives
  - Hybrid solutions
- OS-based DIFC (Asbestos, HiStar, Flume)
  - Run-time tracking enforced by a trusted kernel
    - Works with any language, compiled or scripting, closed or open

Distributed Information Flow Control



Information Flow Control (IFC)

- Goal: track which secrets a process has seen
- Mechanism: each process gets a *secrecy label*
  - Label summarizes which categories of data a process is assumed to have s
  - Examples:
    - { “Financial Reports” }
    - { “HR Documents” }
    - { “Financial Reports” *and* “HR Documents” }

Tags + Labels

Process *p*

$S_p = \{ \}$

$D_p = \{ \}$

```
change_label({Finance});
tag t = create_tag();
change_l
change_l
```

DIFC Rule: A process can create a new tag; gets ability to declassify it.

in action.

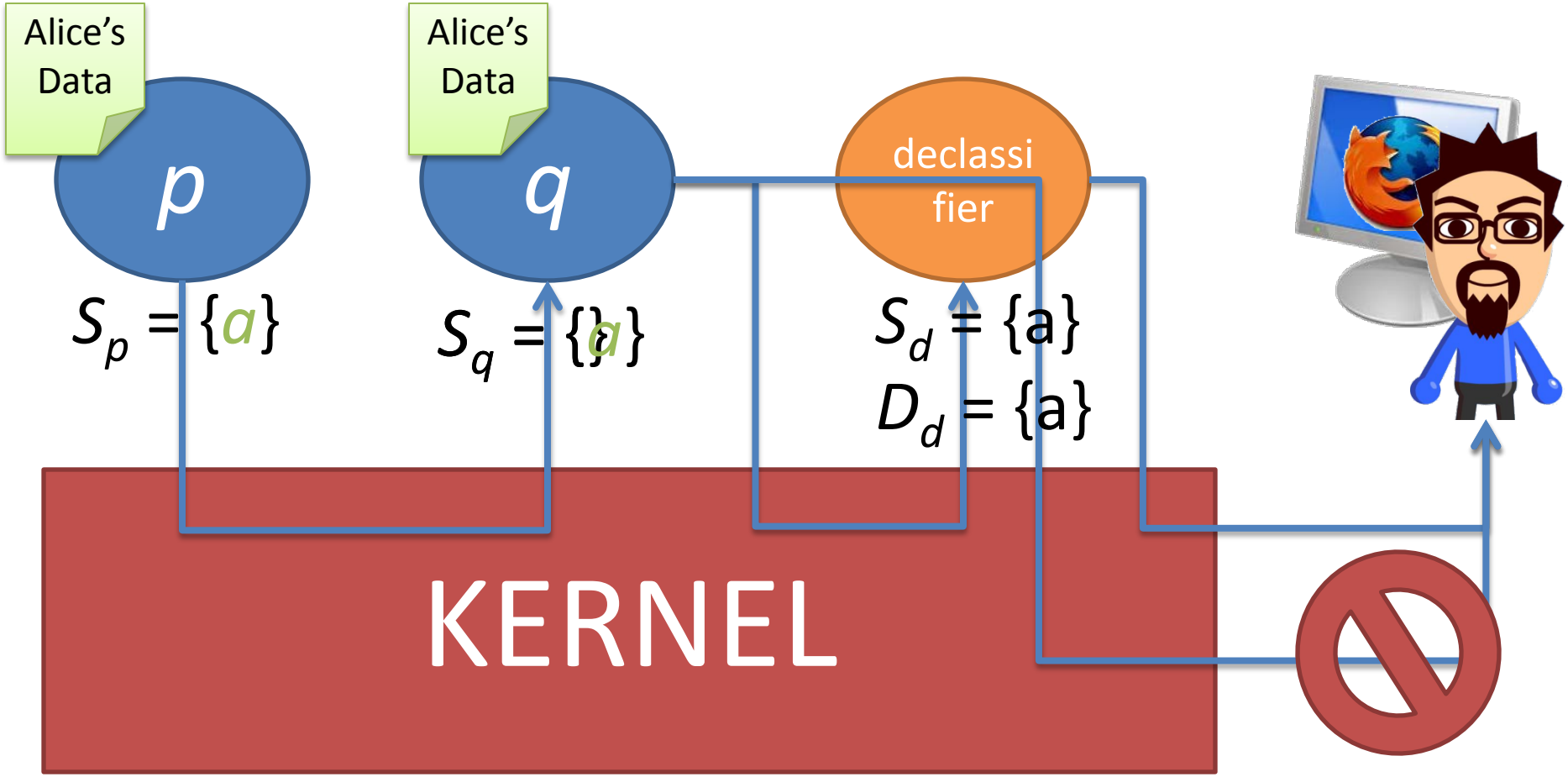
Universe of Tags:

HR

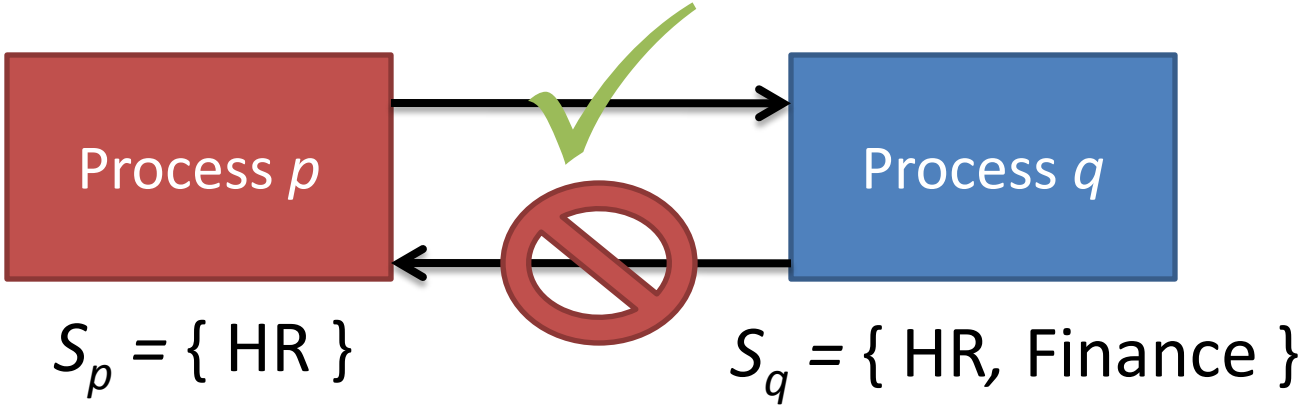
Finance

SecretProjects

DIFC OS


Alice's Data
Alice's Data
p
q
declassi fier
$S_p = \{a\}$
$S_q = \{a\}$
$S_d = \{a\}$
$D_d = \{a\}$
KERNEL

Communication Rule



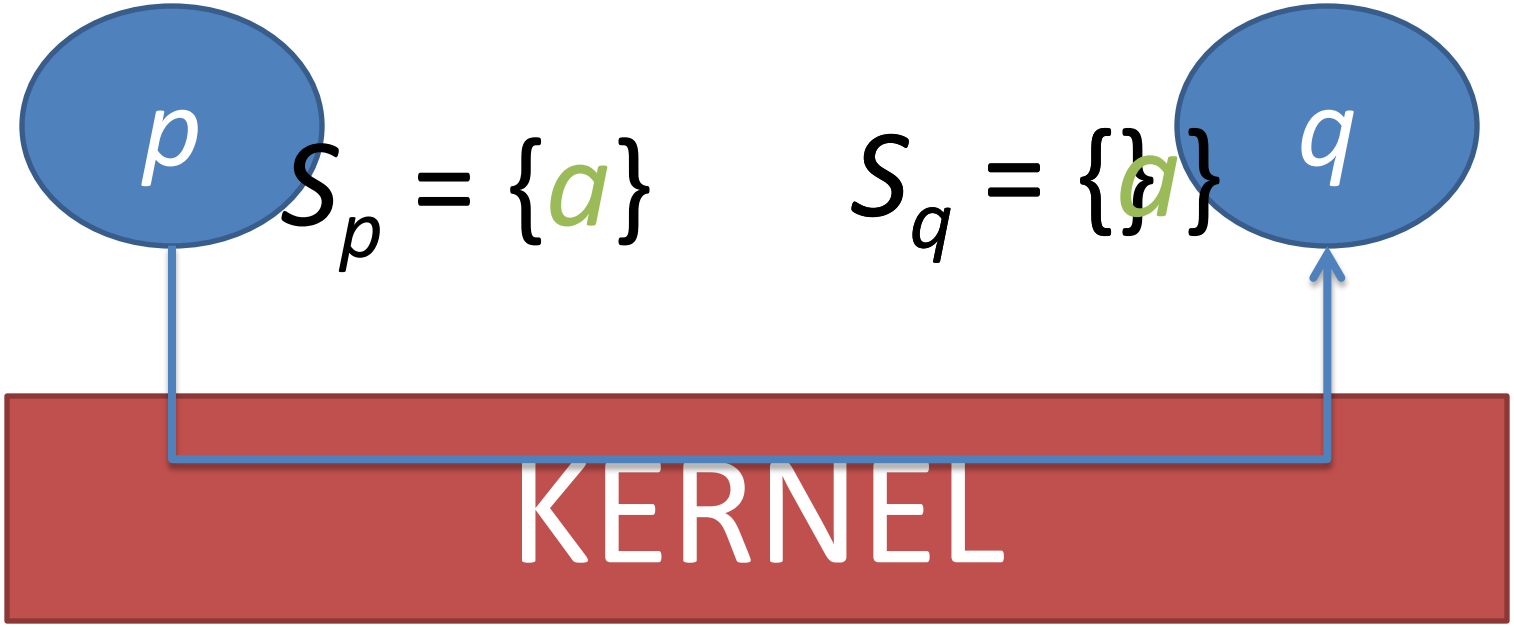
p can send to q iff $S_p \subseteq S_q$

Question:

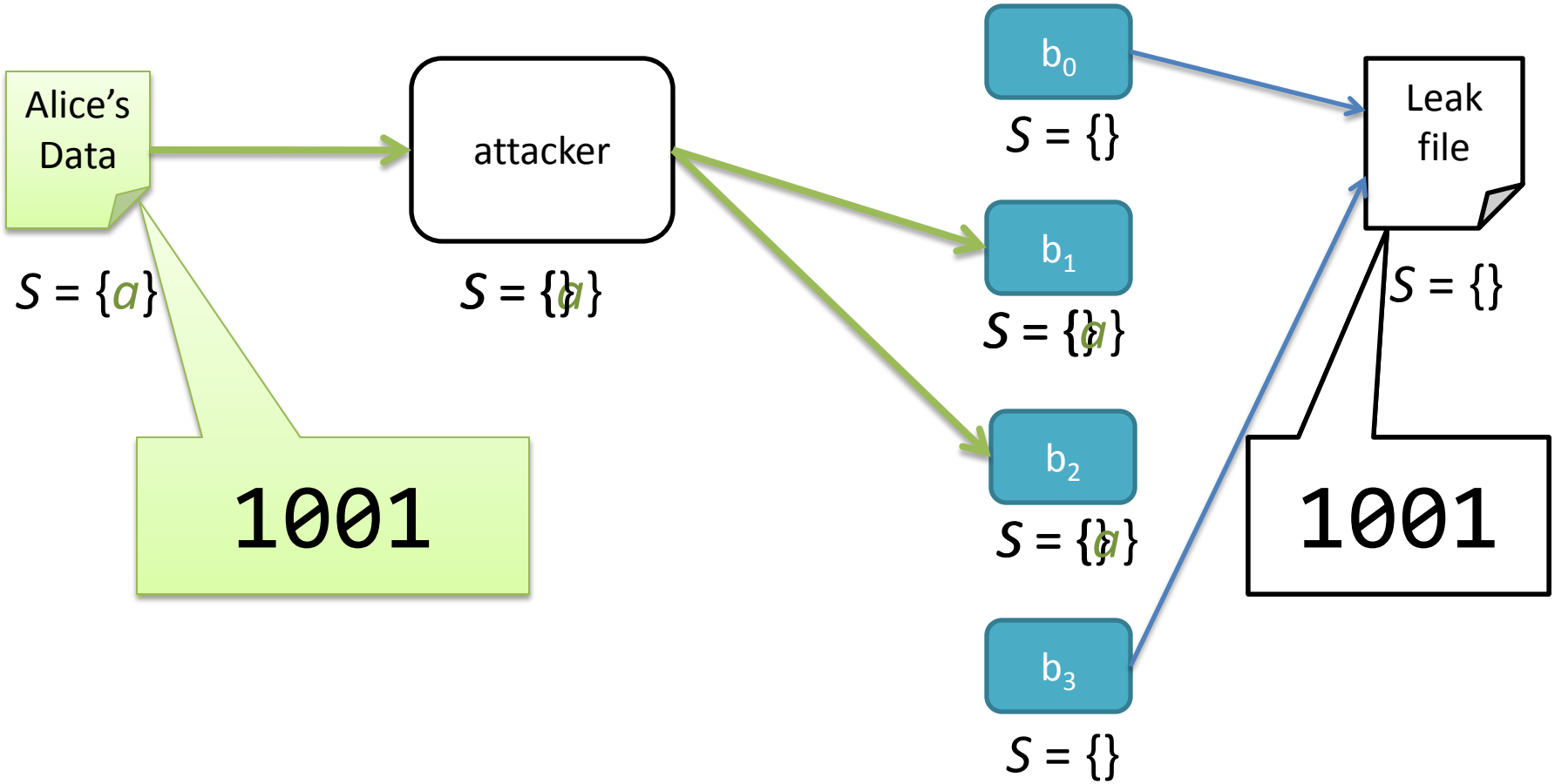
- What interface should the kernel expose to applications?
  - Easier question than “is my OS implementation secure”
  - Easy to get it wrong!

Approach 1: "Floating Labels"

[IX,Asbestos]

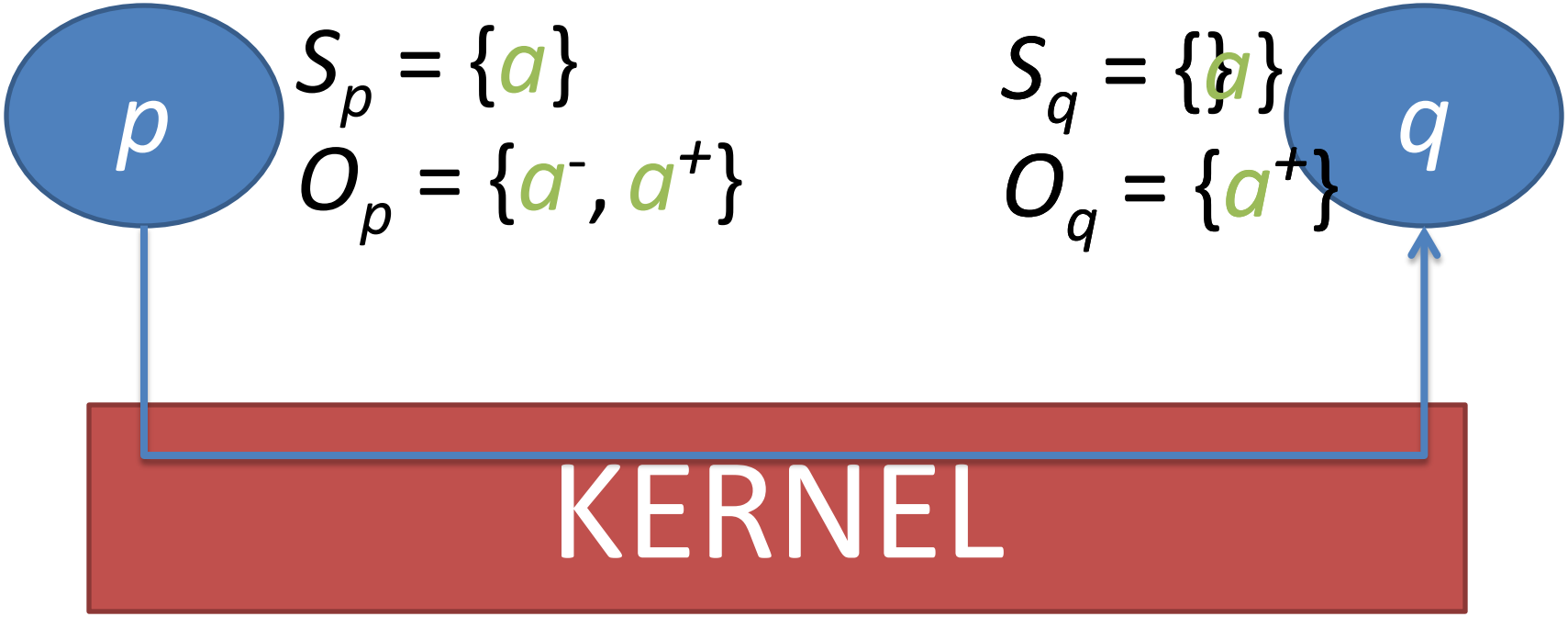


Floaters Leaks Data



Approach 2: "Set Your Own"

[HiStar/Flume]



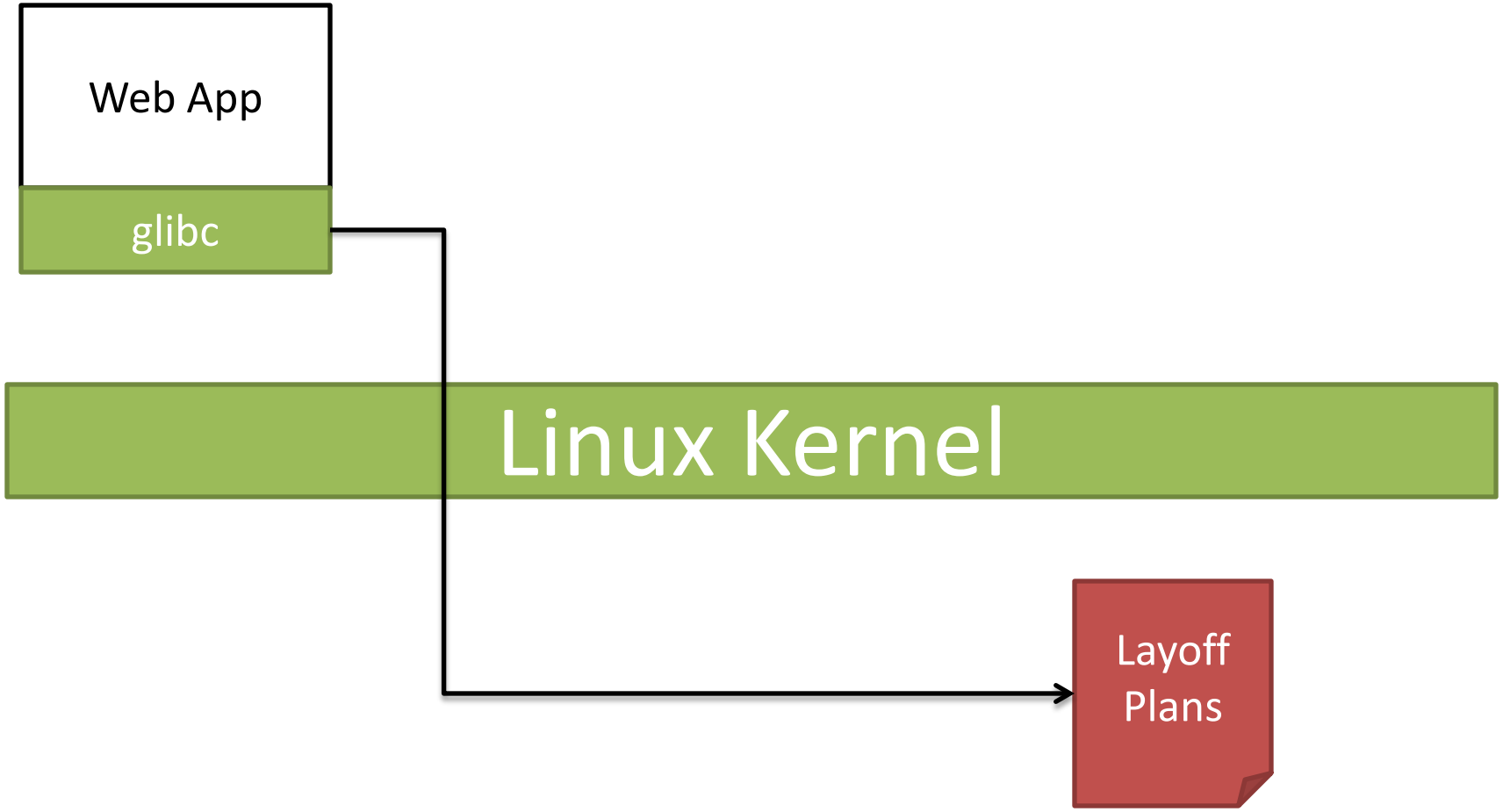
Rule: $S_p \subseteq S_q$ necessary precondition for send

Case study: Flume

- Goal: User-level implementation
  - `apt-get install flume`
- Approach:
  - System Call Delegation [*Ostia* by Garfinkel et al, 2003]
  - Use Linux 2.6 (or OpenBSD 3.9)

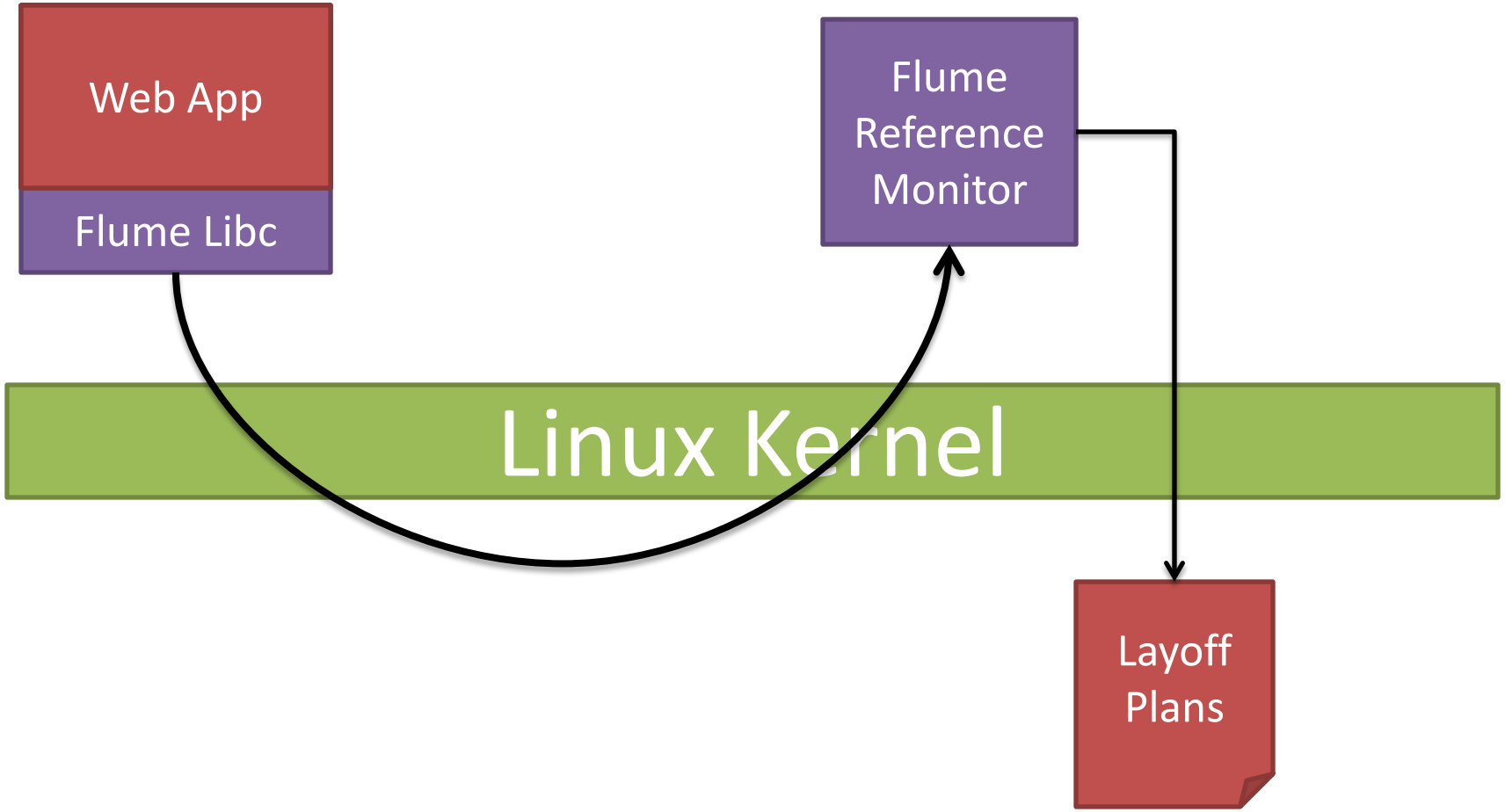
System Call Delegation

`open("/hr/LayoffPlans", O_RDONLY);`



System Call Delegation

`open("/hr/LayoffPlans", O_RDONLY);`



System Calls

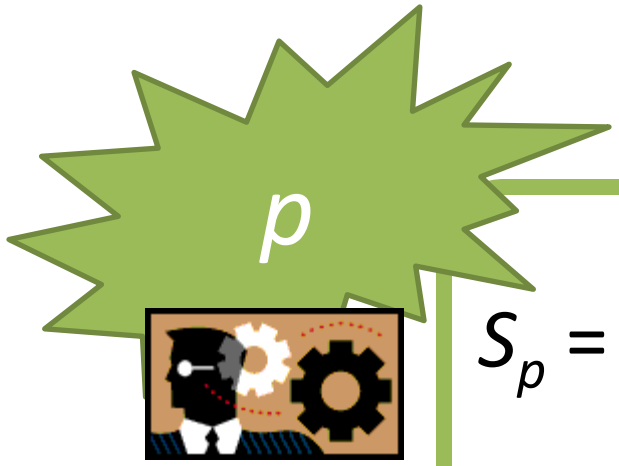
- IPC: read, write, select
- Process management: fork, exit, getpid
- DIFC: create tag, change label, fetch label, send capabilities, receive capabilities

How To Prove Security

- Property: Noninterference

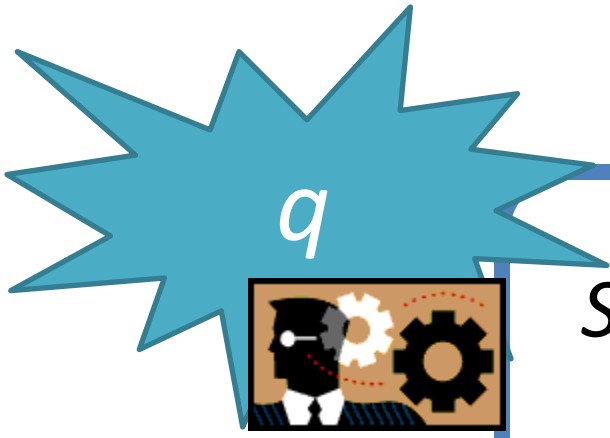
Noninterference [Goguen & Meseguer '82]

Experiment #1:



$S_p = \{a\}$

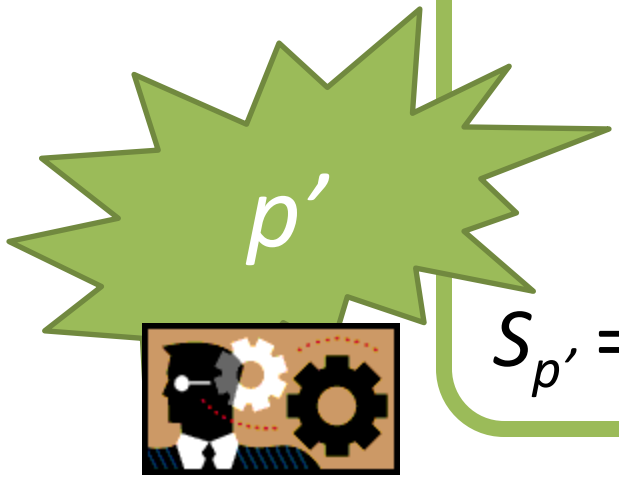
"HIGH"



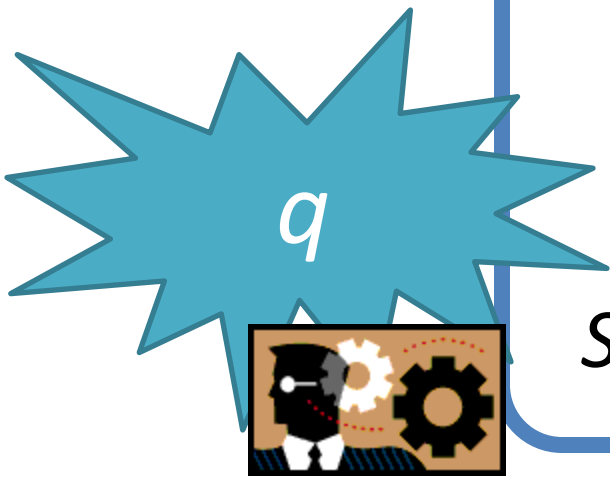
$S_q = \{\}$

"LOW"

Experiment #2:



$S_{p'} = \{a\}$



$S_q = \{\}$

How To Prove Security (cont.)

- Property: Noninterference
- Process Algebra model for a DIFC OS
  - Communicating Sequential Processes (CSP)
- Proof that the model fits the definition

Proving noninterference

- Formalism (Communicating Sequential Formalisms)
- Consider any possible *system* (with arbitrary user applications)
- Induction over all possible sequences of moves a system can make (i.e., *traces*)
- At each step, case-by-case analysis of all system calls in the interface.
  - Prove no “interference”

Results on Flume

- Allows adoption of Unix software?
  - 1,000 LOC launcher/declassifier
  - 1,000 out of 100,000 LOC in MoinMoin changed
  - Python interpreter, Apache, unchanged
- Solves security vulnerabilities?
  - Without our knowing, we inherited two ACL bypass bugs from MoinMoin
  - Both are not exploitable in Flume's MoinMoin
- Performance?
  - Performs within a factor of 2 of the original on read and write benchmarks
- Example App: MoinMoin Wiki

Flume Limitations

- Bigger TCB than HiStar / Asbestos
  - Linux stack (Kernel + glibc + linker)
  - Reference monitor (~22 kLOC)
- Covert channels via disk quotas
- Confined processes like MoinMoin don't get full POSIX API.
  - `spawn()` instead of `fork()` & `exec()`
  - `flume_pipe()` instead of `pipe()`

DIFC vs. Traditional MAC

Traditional MAC	DIFC
Military Systems	Web-based Systems, Desktops
Superior Contraining Users	Developers Protecting Themselves from Bugs
Centralized Declassification Policy	Application-Specific Declassification Policies

What are we trusting?

- Proof of noninterference
- Platform
  - Hardware
  - Virtual machine manager
  - Kernel
- Reference monitor
- Declassifier
  - Human decisions
- Covert channels
- Physical access
- User authentication

Practical barriers to deployment of Information Flow Control systems

- Programming model confuses programmers
- Lack of support:
  - Applications
  - Libraries
  - Operating systems
- People don’t care about security until it's too late

Quantitative information flow control

- Quantify how much information (#bits) is leaking from (secret) inputs to outputs
- Approach: dynamic analysis (extended tainting) followed by flow graph analysis
- Example: battleship online game