Mandatory access control and information flow control

Mandatory access controls and security levels

DAC vs. MAC

Discretionary access control

- This is the familiar case.
  - E.g., the owner of a file can make it accessible to anyone.
- This access control is intrinsically limited in saving principals from themselves.
- It is hard to enforce system-wide security.

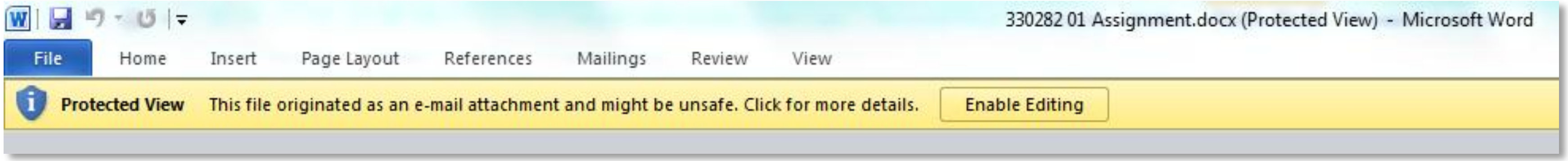
Mandatory access control

- The system assigns security attributes (labels) to both principals and objects.
  - E.g., objects may be “work” or “fun”, and principals may be “trusted” or “guest”.
- These attributes constrain accesses. (Discretionary controls apply in addition.)
  - E.g., “guest” principals cannot modify “work” objects.

MAC (cont.)

- MAC appeared in systems since the 1960s.
- Despite difficulties and disappointments, it also appears more recently, e.g., in
  - *Windows Mandatory Integrity Controls*, where there are four levels for principals and objects:
    - System integrity (e.g., system services)
    - High integrity (e.g., administrative processes)
    - Medium integrity (the default)
    - Low integrity (e.g., for documents from the Internet)
  - *SELinux* “Security-Enhanced Linux” (richer)

A manifestation: protected view of files that arrive by e-mail


330282 01 Assignment.docx (Protected View) - Microsoft Word

Protected View This file originated as an e-mail attachment and might be unsafe. Click for more details.

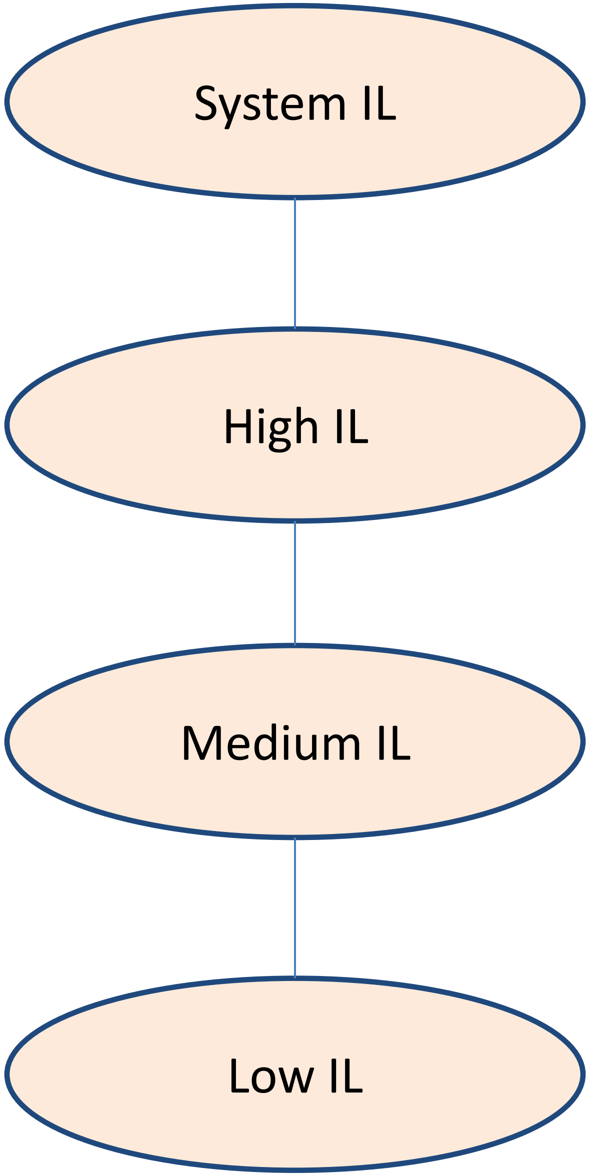
Applications in ***protected mode*** are subject to various restrictions.

Some of these restrictions are achieved by running the applications at Low IL.

Multilevel security

- As in the examples, MAC is often associated with ***security levels***.
- The security levels can pertain to secrecy and integrity properties in a variety of contexts.
- The levels need not be linearly ordered.
  - Often, they form a partial order or a lattice.
  - They may in part reflect a compartment structure.

A partial order



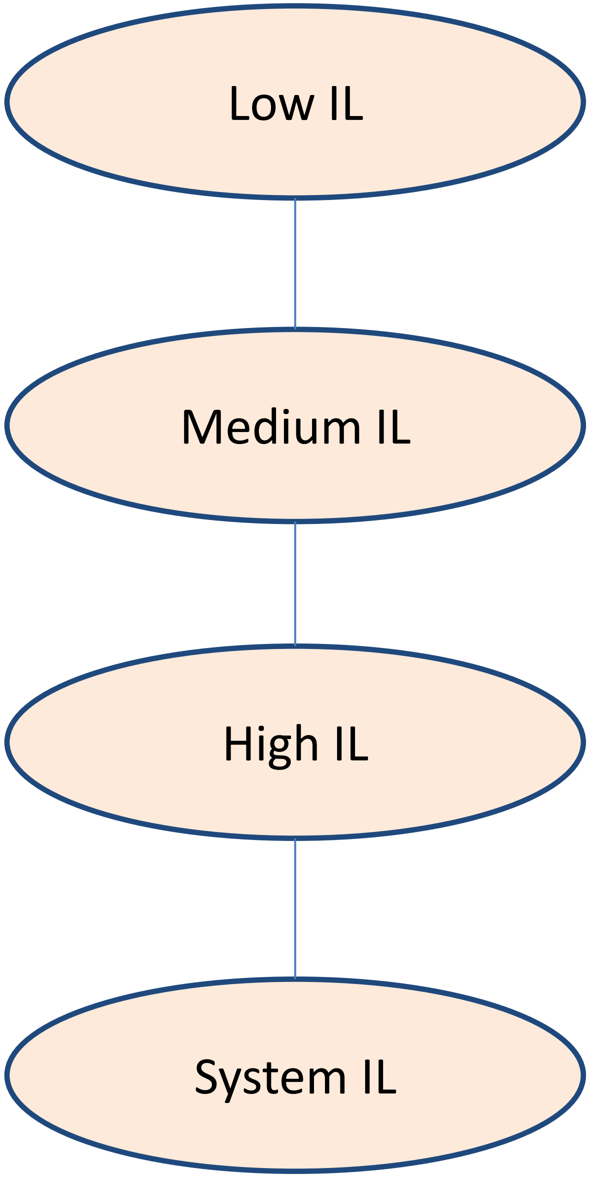
System IL

High IL

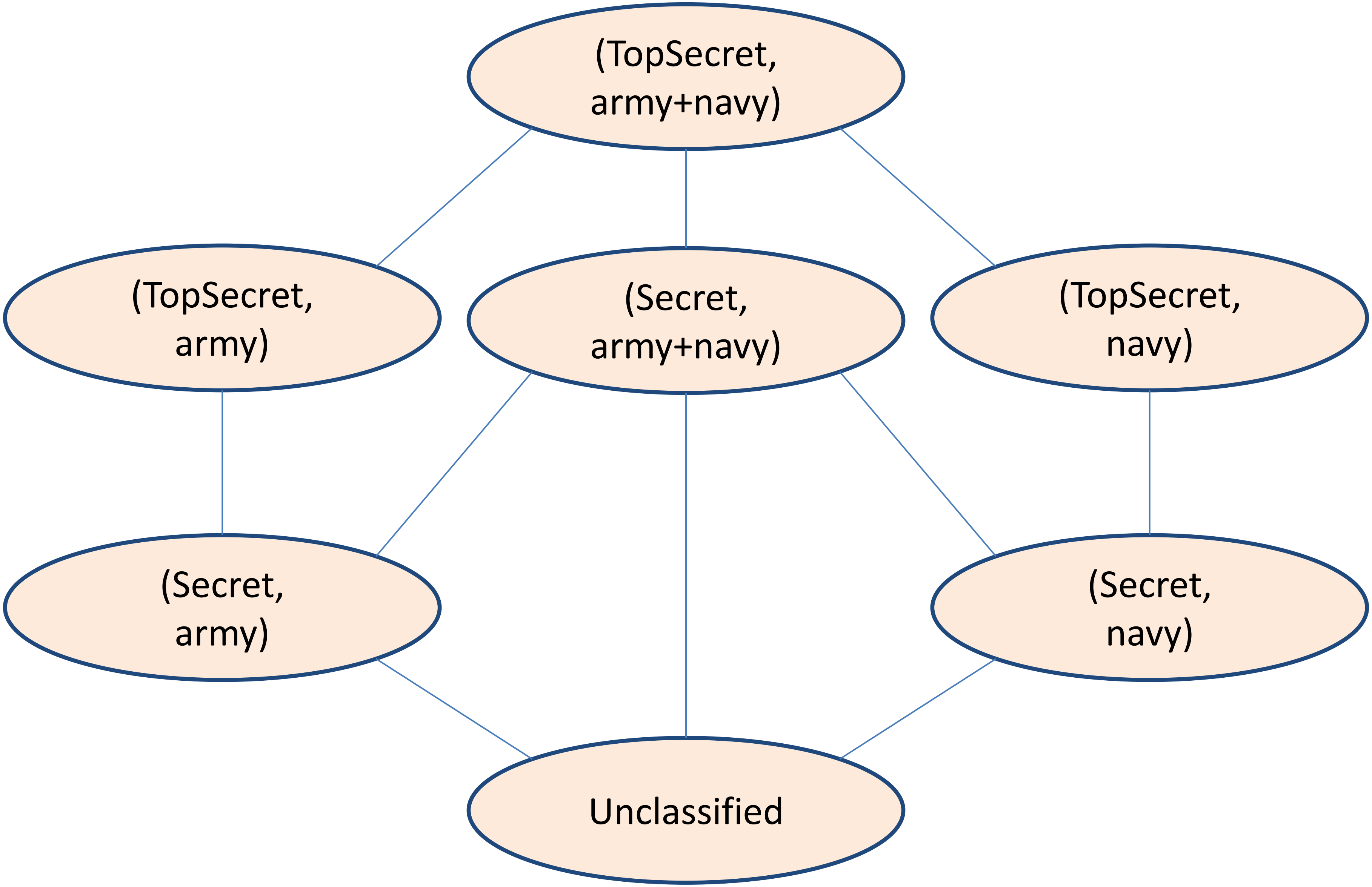
Medium IL

Low IL

Another partial order



Another partial order



(TopSecret, army+navy)

(TopSecret, army)

(Secret, army+navy)

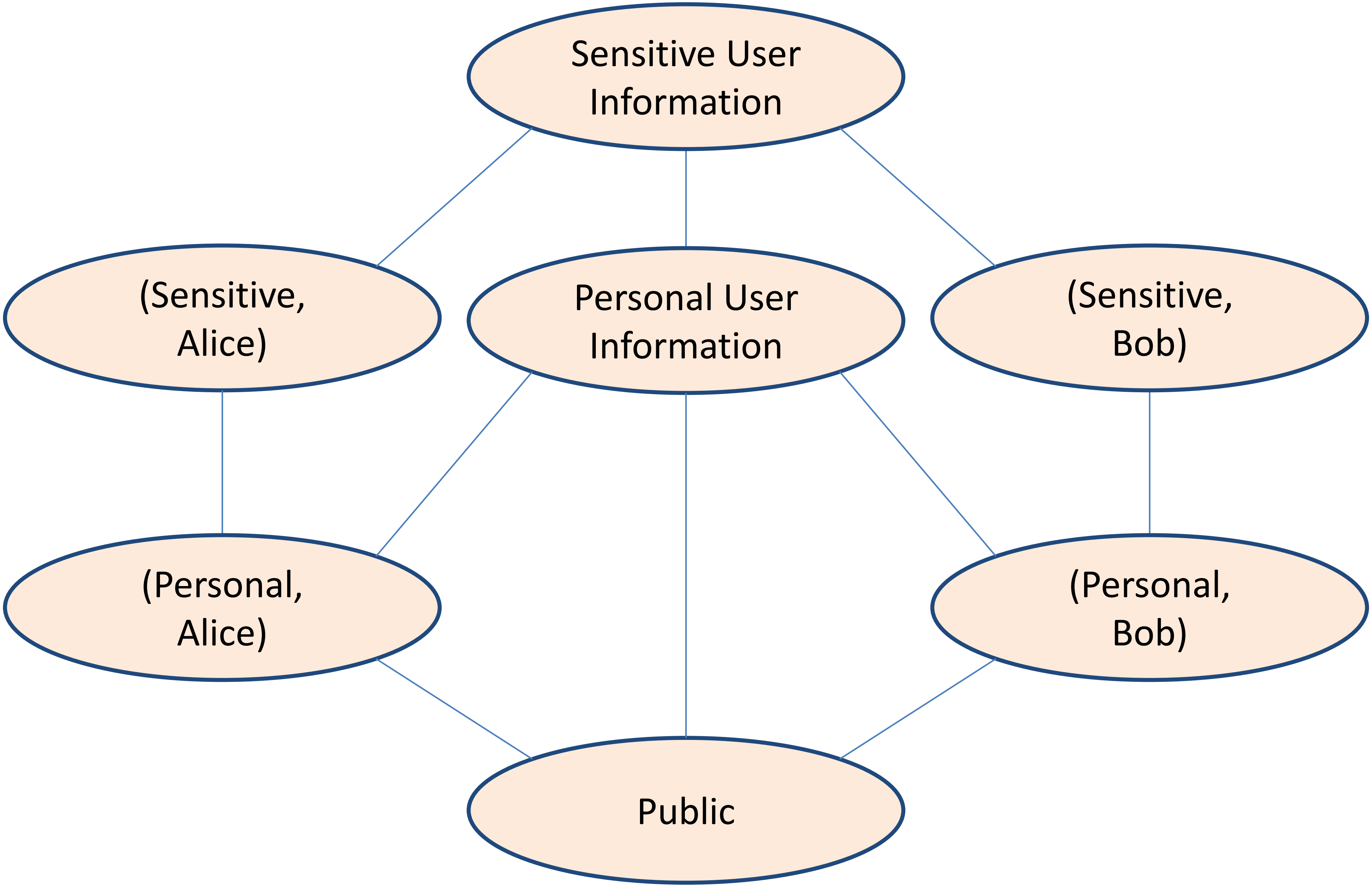
(TopSecret, navy)

(Secret, army)

(Secret, navy)

Unclassified

Another partial order



Bell-LaPadula requirements

- ***No read-up:***
  a principal at a given security level may not read an object at a higher security level.
- ***No write-down:***
  a principal at a given security level may not write to an object at a lower security level.

⇒ protects against **Trojan horses**
(bad programs or other principals that
work at high security levels)

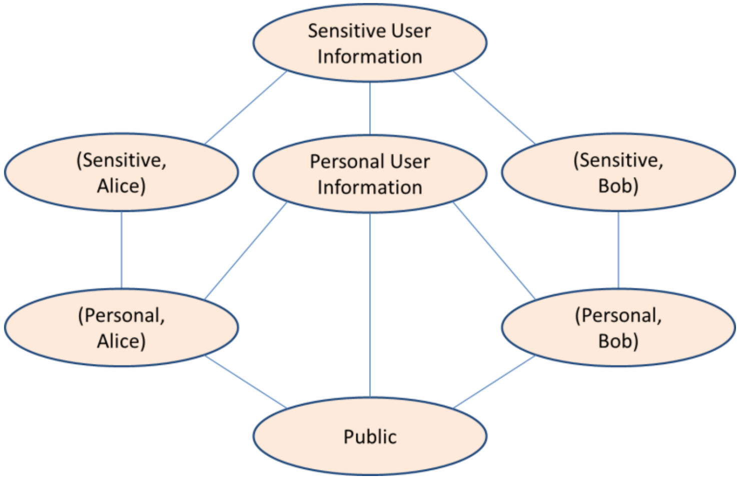
Some difficulties: level creep, declassification, covert channels

Level creep

- Security levels may change over time.
- Security levels tend to creep up.

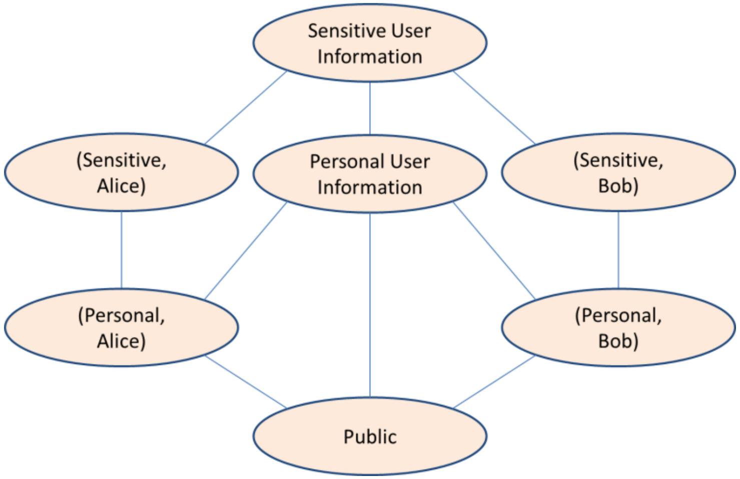
Level creep

- Security levels may change over time.
- Security levels tend to creep up. E.g.:
  – P is a program that may run at any level.
  – blog.html is a file of initial level Public,
  – AliceDiary.txt is a file of initial level (Sensitive, Alice).
  – P may start at Public and write to blog.html, then go to (Sensitive, Alice) and read AliceDiary.txt.
  – Afterwards, P can no longer write to blog.html unless blog.html’s level is raised to (Sensitive, Alice).



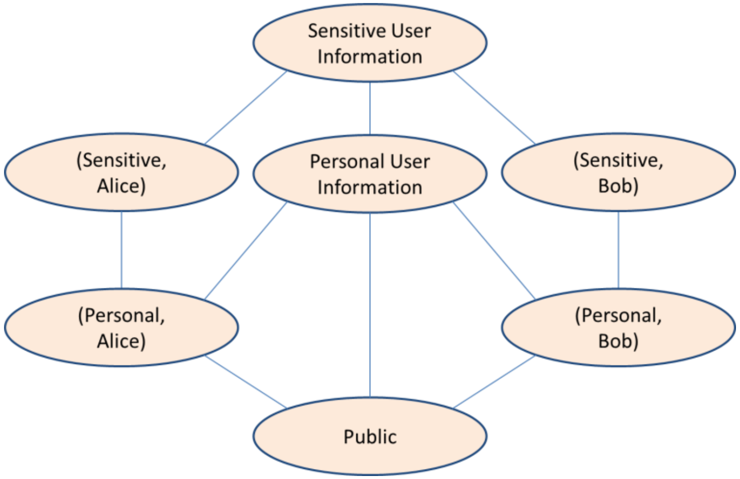
Level creep

- Security levels may change over time.
- Security levels tend to creep up. E.g.:
  - P is a program that may run at any level.
  - blog.html is a file of initial level Public,
  - AliceDiary.txt is a file of initial level (Sensitive, Alice).
  - P may start at Public and write to blog.html, then go to (Sensitive, Alice) and read AliceDiary.txt.
  - Afterwards, P can no longer write to blog.html unless blog.html's level is raised to (Sensitive, Alice).



Level creep

- Security levels may change over time.
- Security levels tend to creep up. E.g.:
  - P is a program that may run at any level.
  - blog.html is a file of initial level Public,
  - AliceDiary.txt is a file of initial level (Sensitive, Alice).
  - P may start at Public and write to blog.html, then go to (Sensitive, Alice) and read AliceDiary.txt.
  - Afterwards, P can no longer write to blog.html unless blog.html's level is raised to (Sensitive, Alice).

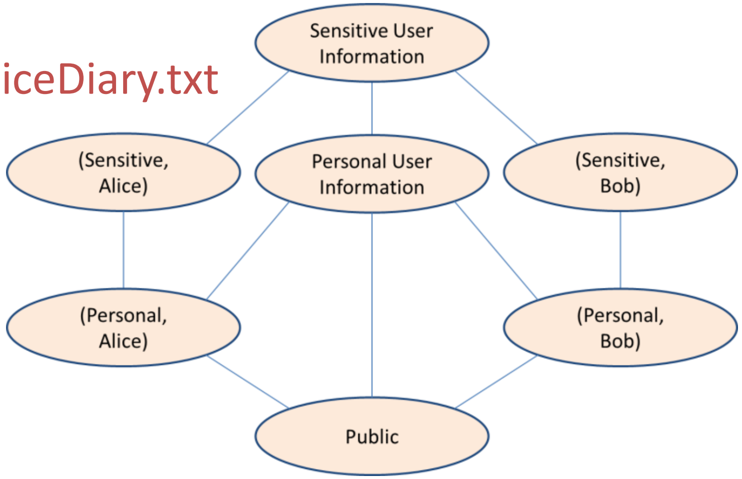


blog.html

Level creep

- Security levels may change over time.
- Security levels tend to creep up. E.g.:
  - P is a program that may run at any level.
  - blog.html is a file of initial level Public,
  - AliceDiary.txt is a file of initial level (Sensitive, Alice).
  - P may start at Public and write to blog.html, then go to (Sensitive, Alice) and read AliceDiary.txt.
  - Afterwards, P can no longer write to blog.html unless blog.html's level is raised to (Sensitive, Alice).

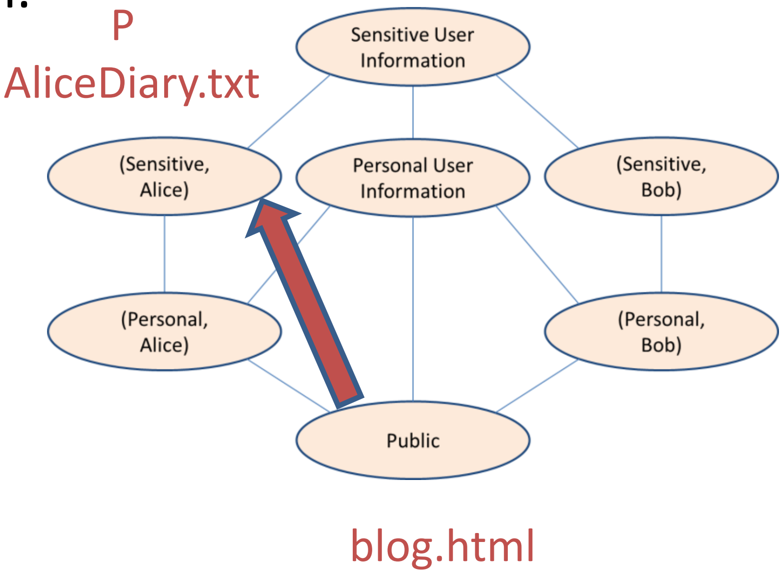
AliceDiary.txt



blog.html

Level creep

- Security levels may change over time.
- Security levels tend to creep up. E.g.:
  - P is a program that may run at any level.
  - blog.html is a file of initial level Public,
  - AliceDiary.txt is a file of initial level (Sensitive, Alice).
  - P may start at Public and write to blog.html, then go to (Sensitive, Alice) and read AliceDiary.txt.
  - Afterwards, P can no longer write to blog.html unless blog.html's level is raised to (Sensitive, Alice).


P

AliceDiary.txt

Sensitive User Information

(Sensitive, Alice)

Personal User Information

(Sensitive, Bob)

(Personal, Alice)

(Personal, Bob)

Public

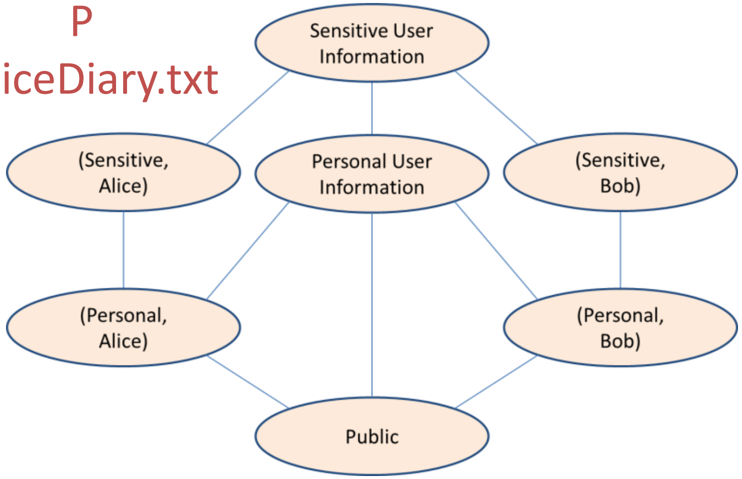
blog.html

Level creep

- Security levels may change over time.
- Security levels tend to creep up. E.g.:
  - P is a program that may run at any level.
  - blog.html is a file of initial level Public,
  - AliceDiary.txt is a file of initial level (Sensitive, Alice).
  - P may start at Public and write to blog.html, then go to (Sensitive, Alice) and read AliceDiary.txt.
  - Afterwards, P can no longer write to blog.html unless blog.html's level is raised to (Sensitive, Alice).

P

AliceDiary.txt



blog.html

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes.
  E.g., the password-checking program reads a secret database and says yes/no to a user.

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes.
  E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult
  - because of hidden messages
    (in text, in pictures, …)

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes. E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult
  - because of hidden messages (in text, in pictures, …)

A boat, beneath a sunny sky
Lingering onward dreamily
In an evening of July -
Children three that nestle near,
Eager eye and willing ear,

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes. E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult
  - because of hidden messages (in text, in pictures, …)

A
L
I
C
E

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes.
  E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult
  - because of hidden messages
    (in text, in pictures, …)

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes.
  E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult
  - because of hidden messages
    (in text, in pictures, …)

Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes. E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult
  - because even the act of declassification may reveal some information

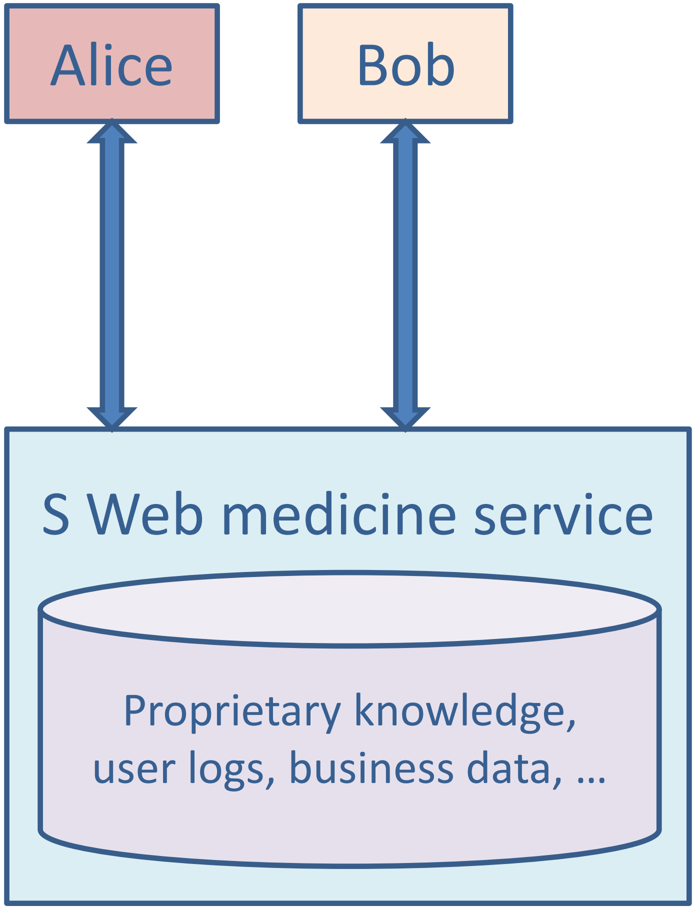
Declassification

- Reclassification consists in changing the security attributes. Declassification is the case in which this is not automatically ok.
- Declassification is needed sometimes.
  E.g., the password-checking program reads a secret database and says yes/no to a user.
- It is difficult.

⇒ *It is a special process, often manual.*

Mutual distrust

Consider a Web service S that offers information to users (e.g., advice or ads).
S relies on proprietary information and user data (e.g., financial data, preferences, email, clicks).
What is a reasonable policy?
Who can declassify what?



Covert channels

Covert channels are communication channels for which the model does not account and which were not intended for communication.

E.g., programs may communicate by the use of shared resources:

By varying its ratio of computing to input/output or its paging rate, the service can transmit information which a concurrently running process can receive by observing the performance of the system.

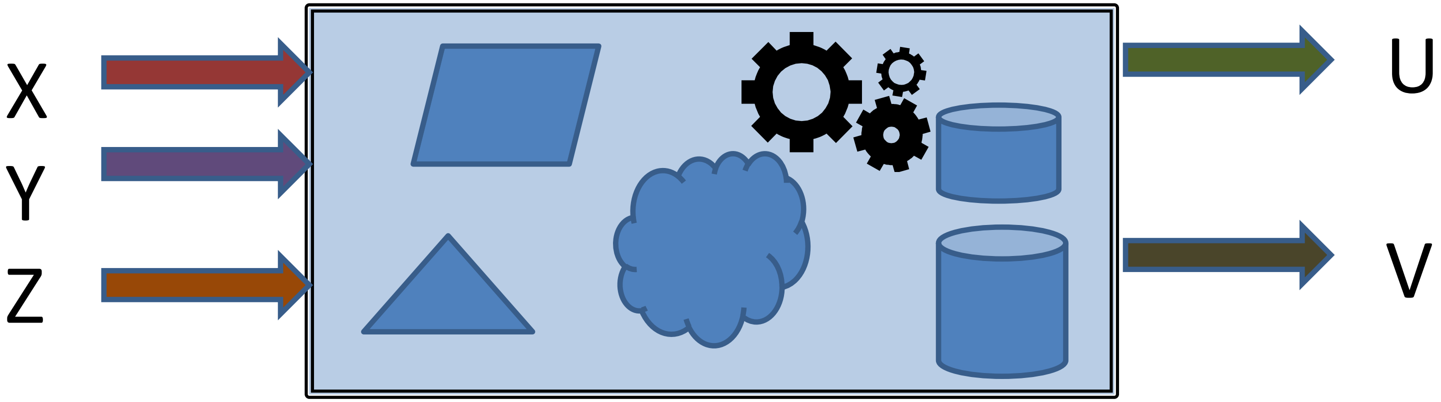
Lampson, 1973

- The “service” may be a Trojan horse, without network access.
- The “concurrently running process” may have a lower level and send any information that it receives on the network.

Information flow

Information flow security

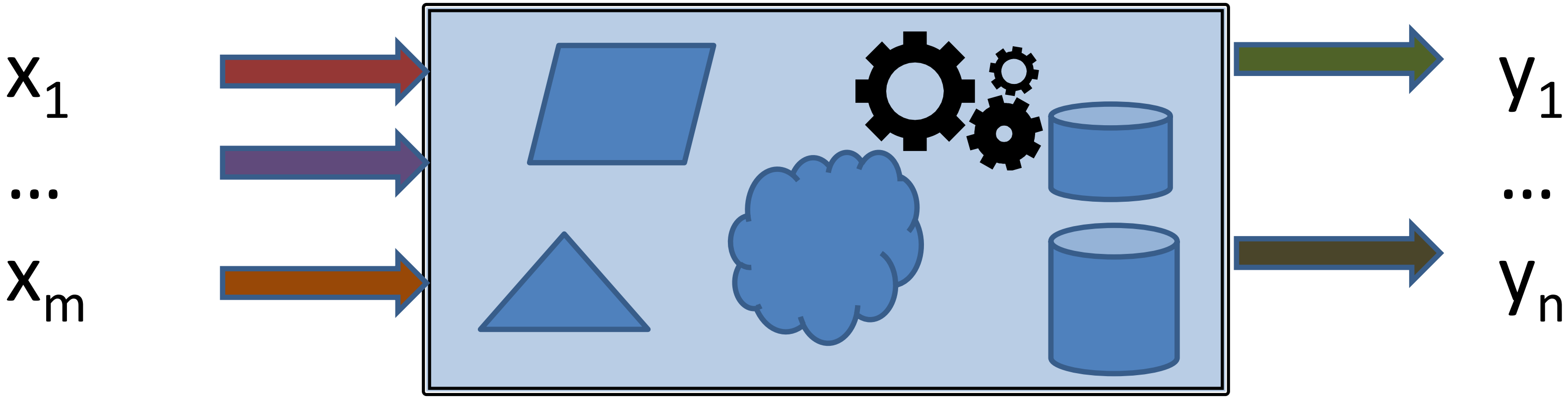
- Access control, of any kind, is limited to the defined principals, objects, and operations.
- Instead, information flow control focuses on the information being protected, *end-to-end*.



E.g., we may want that U do not depend on Z, that is, that Z does not *interfere* with U.

Noninterference: preliminaries

- Consider a system with inputs $x_1,\ldots,x_m$ and outputs $y_1,\ldots,y_n$.
- Suppose $y_j = f_j(x_1,\ldots,x_m)$.
  - Extensions deal with infinite computations, probabilities, nondeterminism, and more.



Noninterference: independence

- So, suppose $y_j = f_j(x_1,\ldots,x_m)$.
- Then y_j *does not depend* on x_i if, always (for all actual values for the inputs), $f_j(v_1, \ldots, v_i, \ldots, v_m) = f_j(v_1, \ldots, v_i', \ldots, v_m)$.

$\Rightarrow$ Secrecy: the value of y_j reveals nothing about the value of x_i.

$\Rightarrow$ Integrity: the value of y_j is not affected by corruptions in the value of x_i.

Noninterference

- Pick some levels (e.g., Public, TopSecret, etc.) with an order on the levels.
- Assign a level to each input $x_1,\ldots,x_m$ and to each output $y_1,\ldots,y_n$.
- ***Noninterference***:
  An output may depend on inputs of the same level, or lower levels, but not on other inputs.
  - So, e.g., outputs of level Public must not depend on inputs of level Sensitive User Information.

Simple examples

- Suppose that Public $<$ Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.

Simple examples

- Suppose that Public $<$ Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$

Simple examples

- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)

Simple examples

- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)
- $y_1 = x_2$

Simple examples

- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)
- $y_1 = x_2$ *not ok*

  There is an ***explicit flow*** of information.

Simple examples

- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)
- $y_1 = x_2$ *not ok*

  There is an ***explicit flow*** of information.
- if x_2 is odd then $y_1 = 1$ else $y_1 = 0$

Simple examples

- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)
- $y_1 = x_2$ *not ok*

  There is an ***explicit flow*** of information.
- if x_2 is odd then $y_1 = 1$ else $y_1 = 0$ *not ok*

  There is an ***implicit flow*** of information.

Simple examples

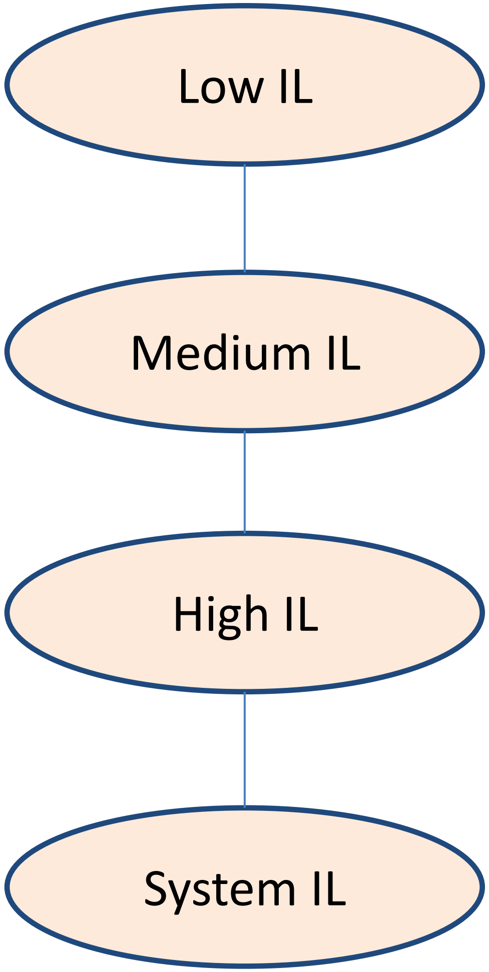
- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)
- $y_1 = x_2$ *not ok*
  There is an ***explicit flow*** of information.
- if x_2 is odd then $y_1 = 1$ else $y_1 = 0$ *not ok*
  There is an ***implicit flow*** of information.
- if x_2 is odd then $y_1 = 1$ else $y_1 = 1$

Simple examples

- Suppose that Public < Secret.
- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok* (y_2 does not matter)
- $y_1 = x_2$ *not ok*
  There is an ***explicit flow*** of information.
- if x_2 is odd then $y_1 = 1$ else $y_1 = 0$ *not ok*
  There is an ***implicit flow*** of information.
- if x_2 is odd then $y_1 = 1$ else $y_1 = 1$ *ok*

Noninterference for integrity

- The definition of noninterference applies to integrity.
- Intuitively the levels need to be ordered “upside-down”.
  - E.g., so that System IL outputs cannot depend on Low IL inputs.



Information flow control for private data?

Information flow control for personal information

- Techniques for detecting the use or release of private data:
  - E.g., for Android apps with TaintDroid [Enck et al.].

  Information flow is typically not wanted.

- Techniques for analysis of private data:
  - In particular, computing aggregates (e.g., number of sick people in a city) without revealing information about individuals (e.g., Alice is sick).

  Some information flow is useful and expected, so relaxed notions of noninterference may be needed.

Approaches to analysis of private data

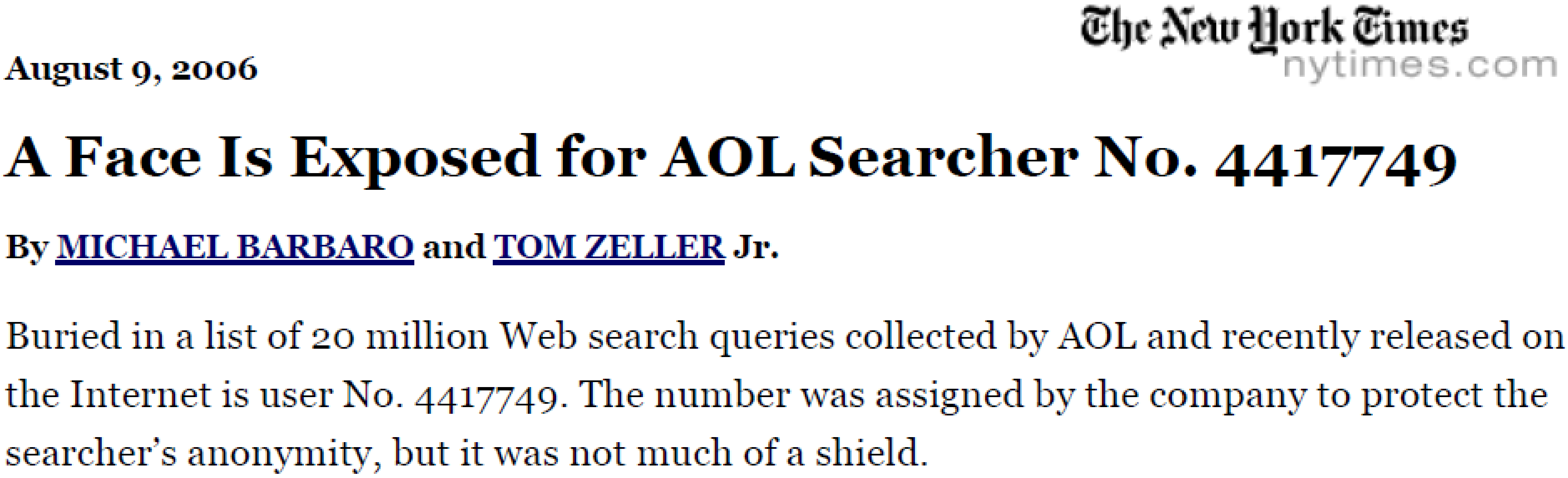
- Anonymizing data.
- Restricting queries.
- Adding noise to input data or to output.

Often ad hoc, sometimes ineffective.

Approaches to analysis of private data

- Anonymizing data.
- Restricting queries.
- Adding noise to input data or to output.

Often ad hoc, sometimes ineffective.



The New York Times
nytimes.com

August 9, 2006

A Face Is Exposed for AOL Searcher No. 4417749

By MICHAEL BARBARO and TOM ZELLER Jr.

Buried in a list of 20 million Web search queries collected by AOL and recently released on the Internet is user No. 4417749. The number was assigned by the company to protect the searcher's anonymity, but it was not much of a shield.

A framework for adding noise to outputs [Dwork, McSherry, Nissim, and Smith]

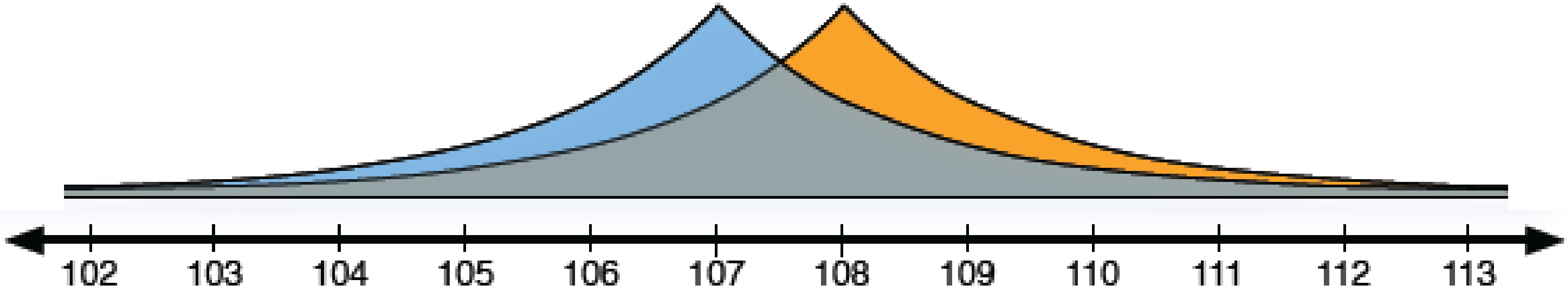
- Algorithm *K* (e.g., counting of sick people) gives ***ε differential privacy*** if, for all **DB** and **DB’** that differ in at most one record, for all v, Prob[*K*(**DB**) = v] ≤ Prob[*K*(**DB’**) = v] × e^{ε}

A framework for adding noise to outputs [Dwork, McSherry, Nissim, and Smith]

- Algorithm *K* (e.g., counting of sick people) gives ε ***differential privacy*** if, for all **DB** and **DB'** that differ in at most one record, for all v, $\mathrm{Prob}[K(\mathbf{DB}) = v] \leq \mathrm{Prob}[K(\mathbf{DB'}) = v] \times e^{\varepsilon}$
- Differential privacy can be achieved by adding noise to outputs.

A framework for adding noise to outputs [Dwork, McSherry, Nissim, and Smith]

- Algorithm *K* (e.g., counting of sick people) gives ***ε differential privacy*** if, for all **DB** and **DB’** that differ in at most one record, for all v, Prob[K(**DB**) = v] ≤ Prob[K(**DB’**) = v] × e^{ε}
- Differential privacy can be achieved by adding noise to outputs.



Source: F. McSherry

Laplace distribution: probability density at x proportional to $e^{-\varepsilon|x|}$.

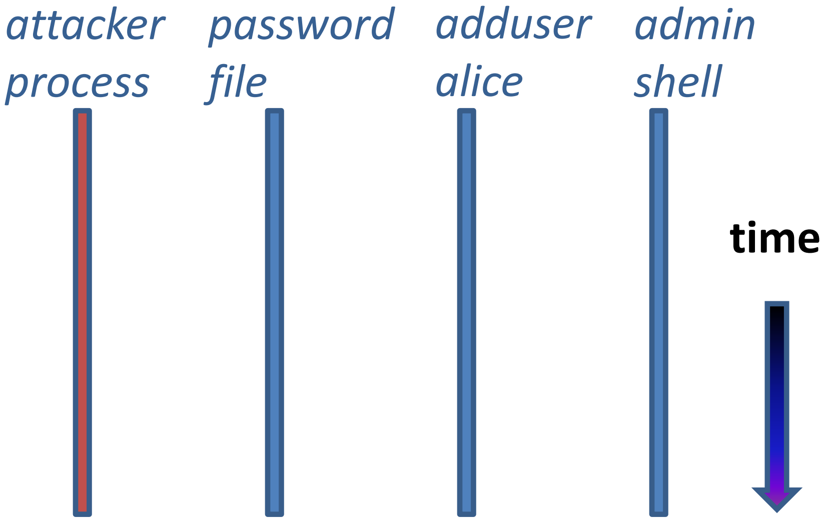
Dynamic information flow control

Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in "undo computing" (loosely) [Kim et al.].

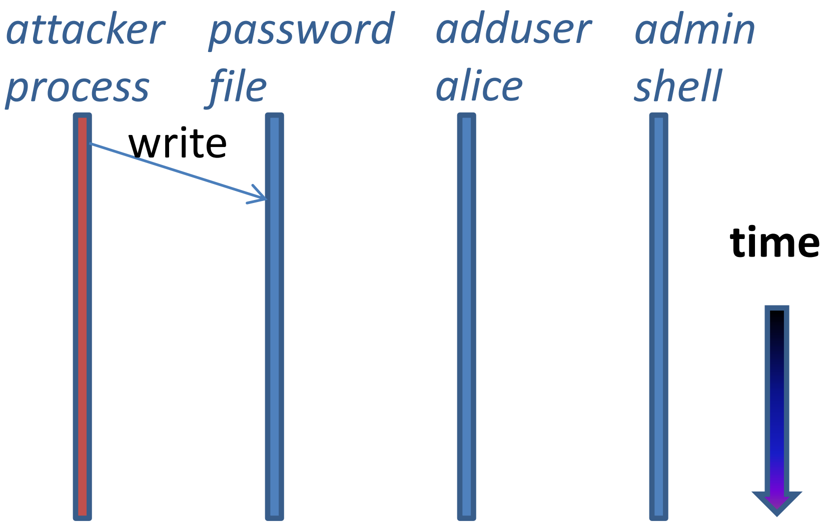
Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in “undo computing” (loosely) [Kim et al.].



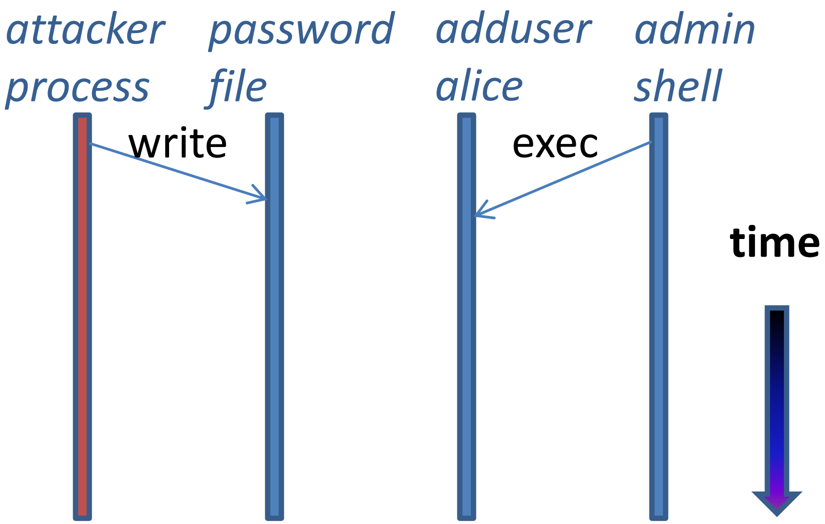
Dynamic information flow control

- Detect information flows dynamically.
  - – A very old idea.
  - – In use, e.g., in Perl taint propagation.
  - – Often expensive, imprecise, but useful.
  - – Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in “undo computing” (loosely) [Kim et al.].



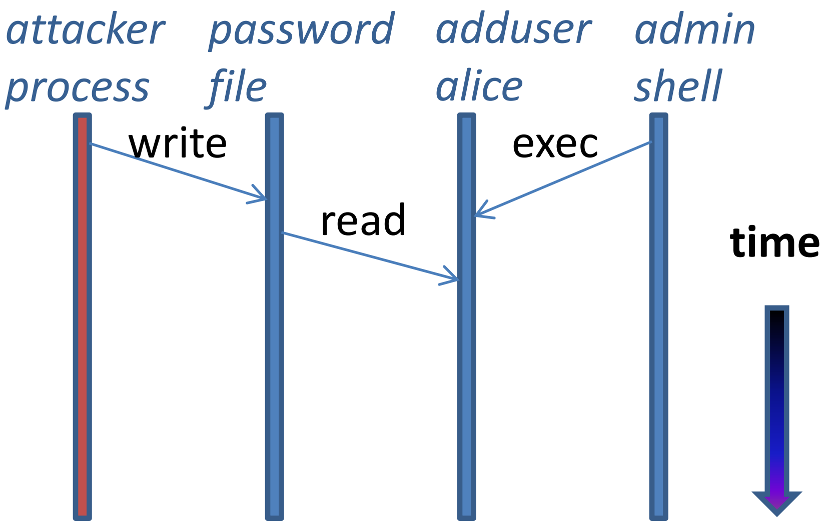
Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in “undo computing” (loosely) [Kim et al.].



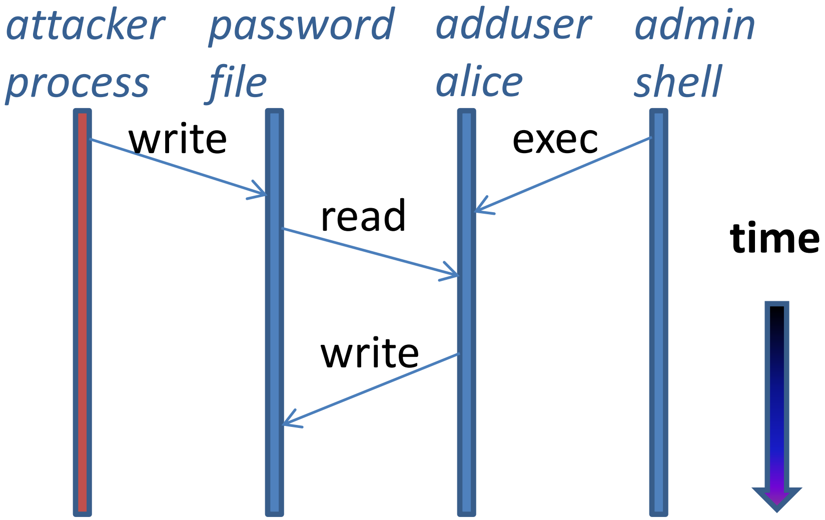
Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in “undo computing” (loosely) [Kim et al.].



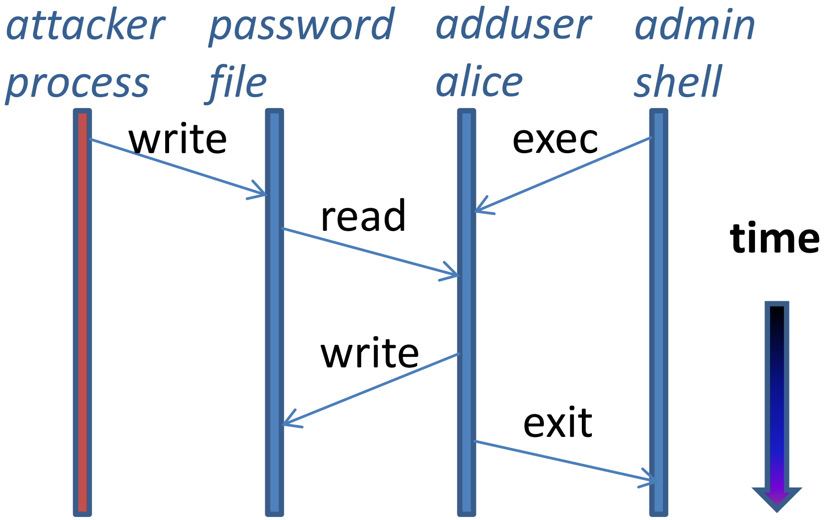
Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in “undo computing” (loosely) [Kim et al.].



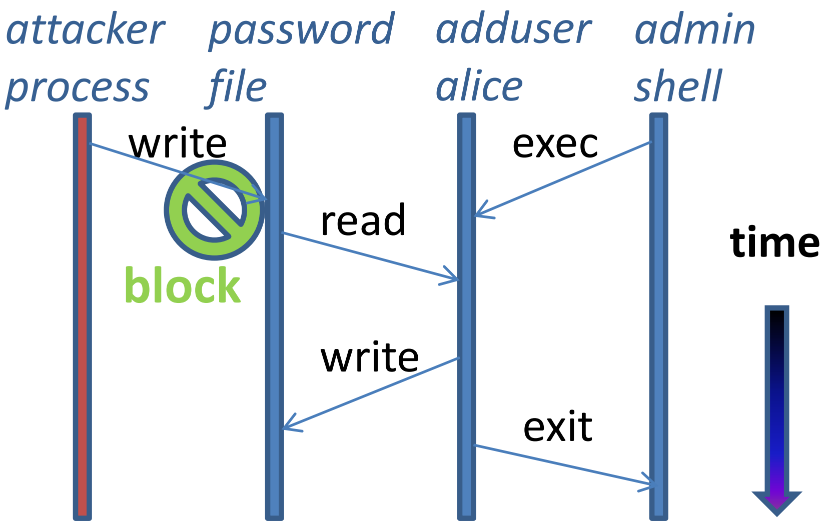
Dynamic information flow control

- Detect information flows dynamically.
  - – A very old idea.
  - – In use, e.g., in Perl taint propagation.
  - – Often expensive, imprecise, but useful.
  - – Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in “undo computing” (loosely) [Kim et al.].



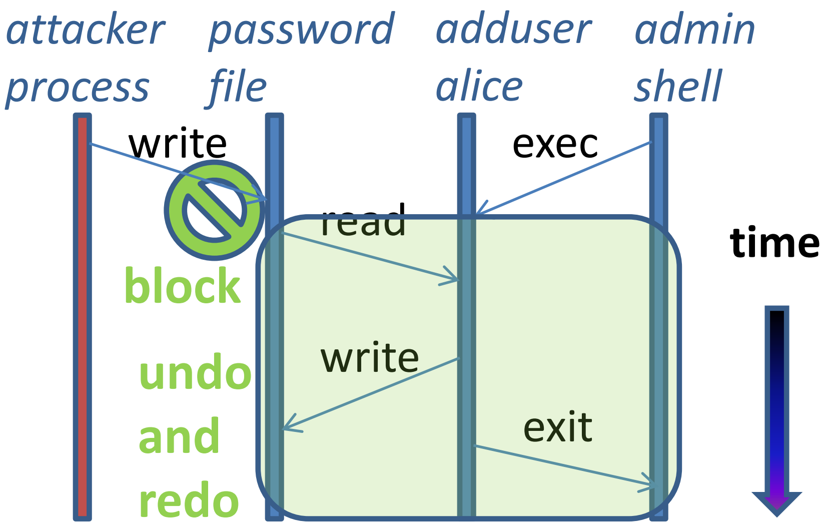
Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in "undo computing" (loosely) [Kim et al.].



Dynamic information flow control

- Detect information flows dynamically.
  - A very old idea.
  - In use, e.g., in Perl taint propagation.
  - Often expensive, imprecise, but useful.
  - Still an active research subject, e.g.,
    - for JavaScript in browsers,
    - in operating systems such Asbestos or HiStar,
    - for selective re-execution in "undo computing" (loosely) [Kim et al.].



Tracking levels: simple examples

Propagate security levels at run-time:

- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok, allowed*
- $y_1 = x_2$ *not ok, easily blocked*
- temp = x_1; y_1 = temp *ok, allowed*
- temp = x_2; y_1 = temp *not ok, easily blocked*

Tracking levels: simple examples

Propagate security levels at run-time:

- Input x_1 and output y_1 have level Public.
- Input x_2 and output y_2 have level Secret.
- $y_1 = x_1$ *ok, allowed*
- $y_1 = x_2$ *not ok, easily blocked*
- temp = x_1; y_1 = temp *ok, allowed*
- temp = x_2; y_1 = temp *not ok, easily blocked*
- if x_2 is odd then $y_1 = 1$ else $y_1 = 0$ *blocked?*
- if x_2 is odd then $y_1 = 1$ else $y_1 = 1$ *blocked?*

A more challenging example

```
$y_2 = x_2$;
$y_1 = 1$;
temp = 1;
if $y_2 = 1$ then temp = 0;
if temp = 1 then $y_1 = 0$;
```

A more challenging example

```
$y_2 = x_2$;
$y_1 = 1$;
temp = 1;
if $y_2 = 1$ then temp = 0;
if temp = 1 then $y_1 = 0$;
```

At the end, if $x_2 = 0$ then $y_1 = 0$, and $y_1 = 1$ otherwise.

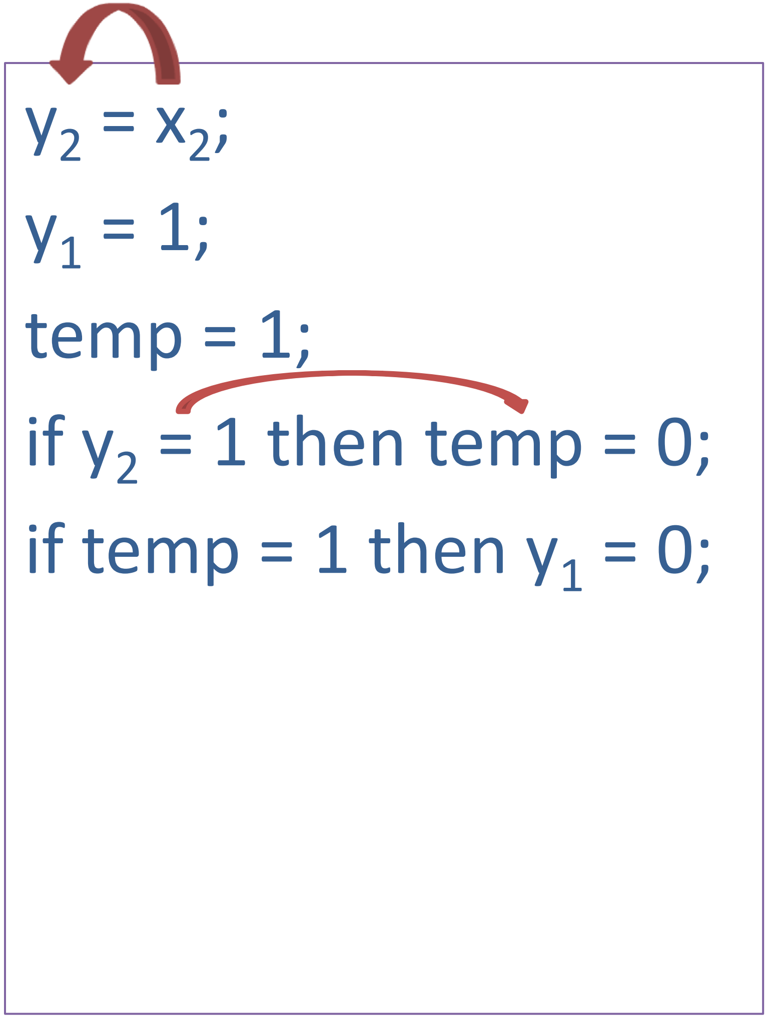
A more challenging example

```
$y_2 = x_2$;
$y_1 = 1$;
temp = 1;
if $y_2 = 1$ then temp = 0;
if temp = 1 then $y_1 = 0$;
```

At the end, if $x_2 = 0$ then $y_1 = 0$, and $y_1 = 1$ otherwise.

$\Rightarrow$ *So there is a flow.*

A more challenging example



```
$y_2 = x_2$;
$y_1 = 1$;
temp = 1;
if $y_2 = 1$ then temp = 0;
if temp = 1 then $y_1 = 0$;
```

At the end, if $x_2 = 0$ then $y_1 = 0$, and $y_1 = 1$ otherwise.

⇒ *So there is a flow.*

When $x_2 = 1$, dynamic taint propagation may suggest that temp is of level Secret.

A more challenging example

```
$y_2 = x_2$;
$y_1 = 1$;
temp = 1;
if $y_2 = 1$ then temp = 0;
if temp = 1 then $y_1 = 0$;
```

At the end, if $x_2 = 0$ then $y_1 = 0$, and $y_1 = 1$ otherwise.

⇒ *So there is a flow.*

When $x_2 = 0$, dynamic taint propagation may suggest that temp is of level Public.

A more challenging example

```
$y_2 = x_2$;
$y_1 = 1$;
temp = 1;
if $y_2 = 1$ then temp = 0;
if temp = 1 then $y_1 = 0$;
```

At the end, if $x_2 = 0$ then $y_1 = 0$, and $y_1 = 1$ otherwise.

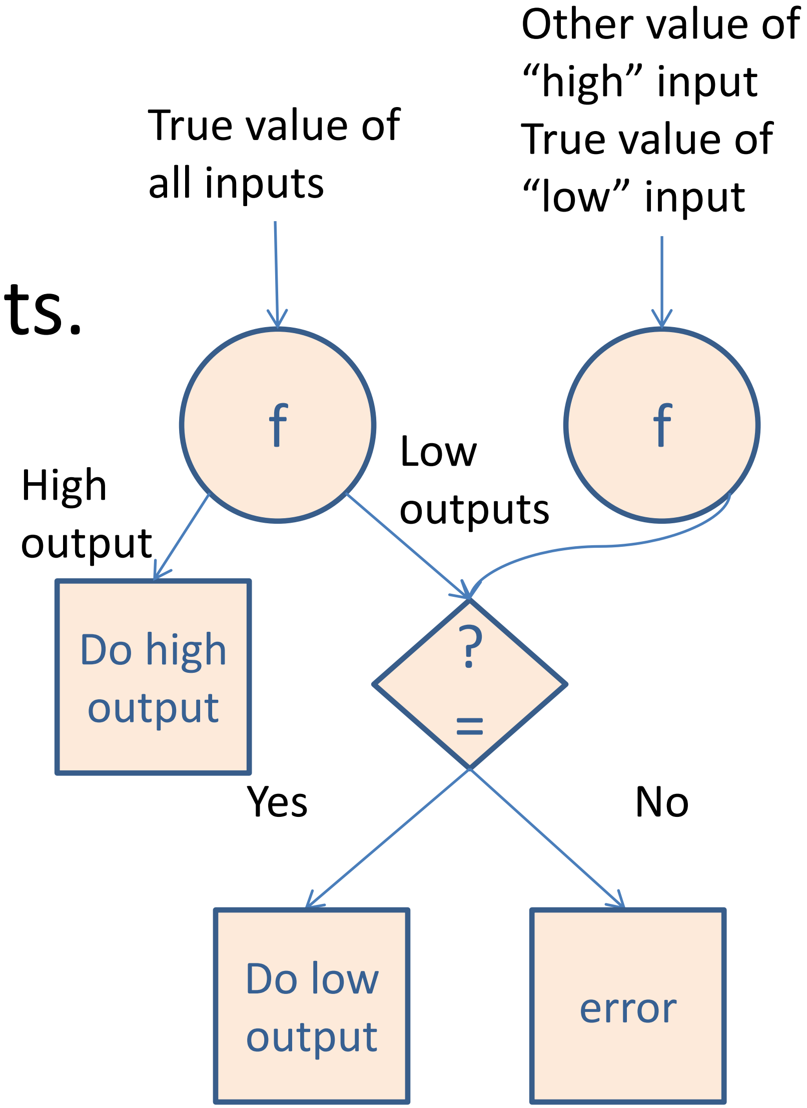
⇒ *So there is a flow.*

In each case, code that is not executed is crucial to the flow!

⇒ *Code analysis is needed.*

Another dynamic technique: multiple executions

- Run multiple copies with different “high” inputs.
- Compare the “low” outputs.
  - If they are equal, then release them.
  - If they are different, then there is information flow, so stop with an error.



True value of all inputs

Other value of “high” input
True value of “low” input

f

f

High output

Low outputs

Do high output

?
=

Yes

No

Do low output

error

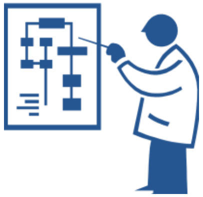
Another dynamic technique: multiple executions (cont.)

- This technique encounters difficulties.
  - – Choice of inputs.
  - – Efficiency of running multiple copies.
  - – Dealing with deliberate nondeterminism.
- But there is research progress.
  - – ML^2 [Simonet and Pottier et al.]
  - – Self-composition [Barthe et al.]
  - – TightLip [Yumerefendi et al.]
  - – Secure Multiexecution [Devriese and Piessens]

Static information flow control

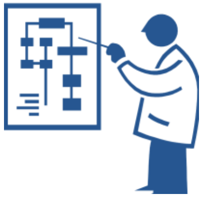
Static information flow control

- Analyze programs before execution.
  - An old idea too.
  - Also with applications to current problems (e.g., finding bugs in Javascript browser extensions with Vex [Bandhakavi et al.]).
  - In recent years, relying on programming-language research (e.g., type systems).



Example of a static approach

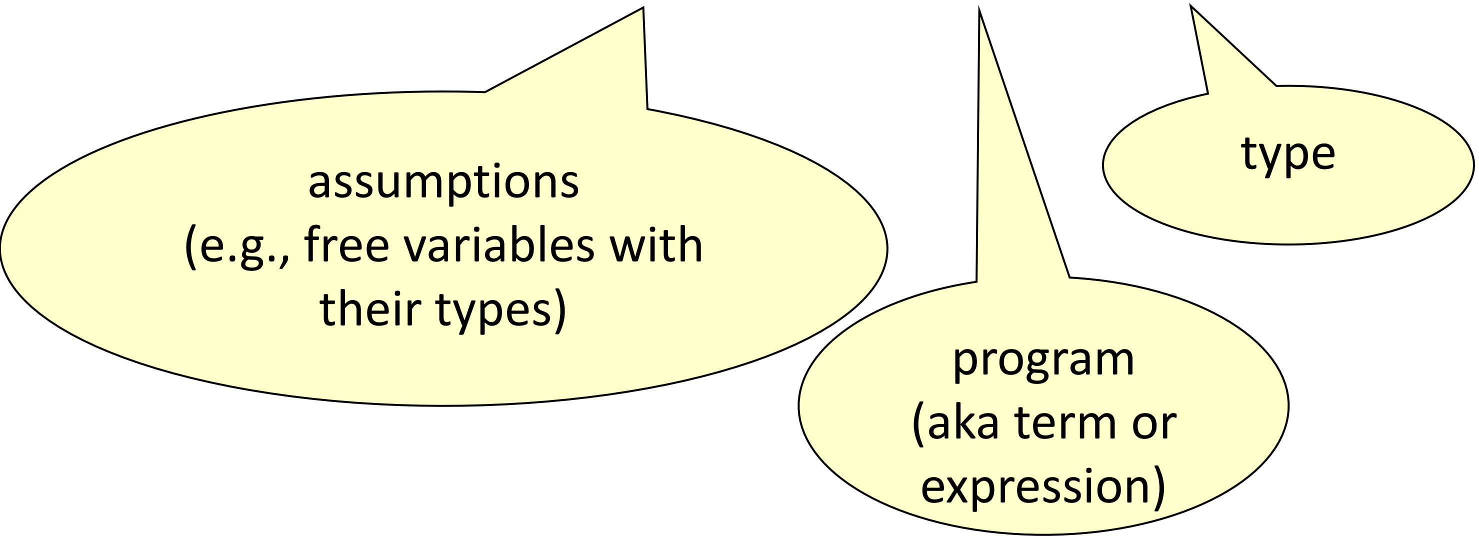
- We treat only simple language constructs (following a “monadic” approach).
- One security level (“High”) is explicit.
  All the rest is implicitly of a “Low” level.
  - E.g., `int` represents the type of (“Low”) integers.
  - High `int` represents the type of High integers (i.e., the secret integers, in one interpretation).
  - `int` outputs should not depend on High `int` inputs.



Typing rules

- As usual, typing rules are rules for deducing judgments (assertions) of the form:

$$\Gamma \vdash e : s$$



assumptions
(e.g., free variables with their types)

program
(aka term or expression)

type

Example judgments and rules

- A judgment: $x : \texttt{int} \vdash x + 0 : \texttt{int}$

- Some rules:

$$\Gamma, x : s, \Gamma' \vdash x : s \qquad\qquad \Gamma \vdash 0 : \texttt{int}$$

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}$$

The Simply Typed λ-calculus: rules

$$\Gamma, x : s, \Gamma' \vdash x : s$$

$$\Gamma \vdash () : \mathtt{true}$$

$$\frac{\Gamma, x : s_1 \vdash e : s_2}{\Gamma \vdash (\lambda x : s_1 .\, e) : (s_1 \to s_2)}$$

$$\frac{\Gamma \vdash e : (s_1 \to s_2) \qquad \Gamma \vdash e' : s_1}{\Gamma \vdash (e\, e') : s_2}$$

$$\frac{\Gamma \vdash e_1 : s_1 \qquad \Gamma \vdash e_2 : s_2}{\Gamma \vdash \langle e_1, e_2 \rangle : (s_1 \times s_2)}$$

$$\frac{\Gamma \vdash e : (s_1 \times s_2)}{\Gamma \vdash (\mathtt{proj}_1\, e) : s_1}$$

$$\frac{\Gamma \vdash e : (s_1 \times s_2)}{\Gamma \vdash (\mathtt{proj}_2\, e) : s_2}$$

$$\frac{\Gamma \vdash e : s_1}{\Gamma \vdash (\mathtt{inj}_1\, e) : (s_1 + s_2)}$$

$$\frac{\Gamma \vdash e : s_2}{\Gamma \vdash (\mathtt{inj}_2\, e) : (s_1 + s_2)}$$

$$\frac{\Gamma \vdash e : (s_1 + s_2) \qquad \Gamma, x : s_1 \vdash e_1 : s \qquad \Gamma, x : s_2 \vdash e_2 : s}{\Gamma \vdash (\mathtt{case}\ e\ \mathtt{of}\ \mathtt{inj}_1(x).\, e_1 \mid \mathtt{inj}_2(x).\, e_2) : s}$$

Rules for High

- High can always be added: $$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta\ e) : \mathtt{High}\ s}$$

Rules for High

- High can always be added: $$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta\ e) : \texttt{High}\ s}$$
  - So for example $(\eta\ 0) : \texttt{High int}$

Rules for High

- High can always be added: $$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta\ e) : \texttt{High}\ s}$$
  - So for example $(\eta\ 0) : \texttt{High int}$
- High expressions can be used in other High expressions: $$\frac{\Gamma \vdash e : \texttt{High}\ s \qquad \Gamma, x : s \vdash e' : \texttt{High}\ t}{\Gamma \vdash \texttt{bind}\ x = e\ \texttt{in}\ e' : \texttt{High}\ t}$$

Rules for High

- High can always be added: $$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta\ e) : \texttt{High}\ s}$$
  - So for example $(\eta\ 0) : \texttt{High int}$
- High expressions can be used in other High expressions: $$\frac{\Gamma \vdash e : \texttt{High}\ s \qquad \Gamma, x : s \vdash e' : \texttt{High}\ t}{\Gamma \vdash \texttt{bind}\ x = e\ \texttt{in}\ e' : \texttt{High}\ t}$$
  - So for example, if $e : \texttt{High int}$ then $\texttt{bind}\ x = e\ \texttt{in}\ (\eta\ (x + 1)) : \texttt{High int}$

Rules for High

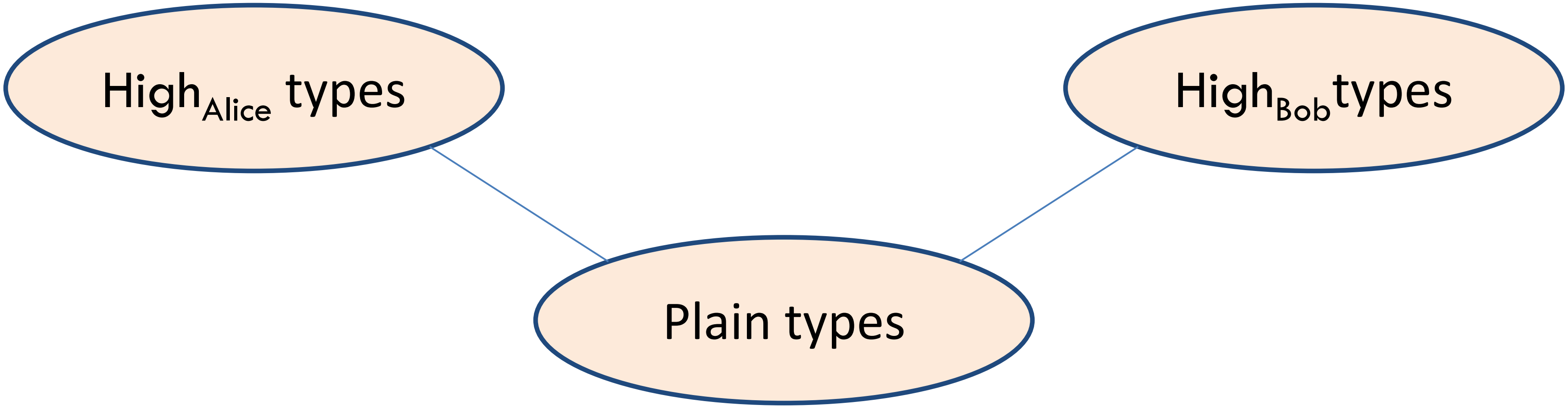
- High can always be added: $$\frac{\Gamma \vdash e : s}{\Gamma \vdash (\eta\ e) : \mathtt{High}\ s}$$
  - So for example $(\eta\ 0) : \mathtt{High\ int}$
- High expressions can be used in other High expressions: $$\frac{\Gamma \vdash e : \mathtt{High}\ s \qquad \Gamma, x : s \vdash e' : \mathtt{High}\ t}{\Gamma \vdash \mathtt{bind}\ x = e\ \mathtt{in}\ e' : \mathtt{High}\ t}$$
  - So for example, if $e : \mathtt{High\ int}$ then $\mathtt{bind}\ x = e\ \mathtt{in}\ (\eta\ (x + 1)) : \mathtt{High\ int}$
  - But there is no way to go from High to Low.

A simple noninterference property

If $\vdash f : (\texttt{High int}) \rightarrow \texttt{int}$,
$\vdash e_1 : \texttt{High int}$, and $\vdash e_2 : \texttt{High int}$,
then $f(e_1)$ and $f(e_2)$ are equal
(that is, evaluate to the same integer).

A first generalization

- Consider multiple principals (Alice, Bob, ...).
- We replace the single level High with a different level High_A for each principal *A*.
  - High_A int may represent the type of *A*’s integer secrets,
  - or the type of integers whose integrity *A* trusts.



$\text{High}_{\text{Alice}}$ types
High_{Bob}types
Plain types

Rules for High$_A$

- The rules are basically those for High:
  - High$_A$ can always be added.
  - High$_A$ expressions can be used in computing other High$_A$ expressions (but there is no way to go from High$_A$ to Low or to High$_B$).

Further work

- Theorems, in particular noninterference.
- More general versions, with more levels, etc..
- Use in languages and systems.
- Connections to access control.

- For richer, more useful and real systems, see in particular Jif [Myers et al.].

Reading

- Sabelfeld & Myers's "Language-Based Information-Flow Security" http://www.cs.cornell.edu/andru/papers/jsac/sm-jsac03.pdf
- Joshi & Leino's "A Semantic Approach to Secure Information Flow" http://rjoshi.org/bio/papers/SCP2000.pdf
- Zeldovich et al.'s "Making Information Flow Explicit in HiStar" http://people.csail.mit.edu/nickolai/papers/zeldovich-histar-cacm.pdf

Homework 3 (due October 25)

Exercise 1: *(on material from earlier lectures)*

Consider a system with three users A, B, and C, two objects O and P, and one operation on the objects.

Consider the following policy (written in Binder notation):

- trusted(A).
- A says trusted(B).
- A says may-access(C,O).
- may-access(x,O) :- trusted(y), y says may-access(x,O).
- may-access(x,P) :- trusted(x).
- trusted(x) :- trusted(y), y says trusted(x).

(x and y are variables that range over A, B, and C.)

Write the resulting access control matrix.

Homework 3

Exercise 2:

The following questions are about the definition of security by Joshi and Leino. If in doubt, you may use the relational version of the definition. For (a)–(g), simply answer “yes”, “no”, or “it depends”.

a) Is HH secure?
b) Is true secure?
c) Is k := k $*$ h secure? ($*$ represents multiplication)
d) Is k := n $*$ h secure, where n is some integer constant?
e) Is h := 0; k := h secure?
f) If S and T are secure, is if k > 0 then S else T end also secure?
g) If S and T are secure, is their composition S; T also secure?
h) Give a proof of your answer to (e) using the relational semantics.
i) Briefly discuss the significance of your answers to (f) and (g).

Homework 3

Exercise 3:

In HiStar (as described in the CACM paper), suppose that there is a thread T and a segment S, and suppose that T can start another thread T' that can read S. Discuss whether it follows that T can also read S

a) in general,

b) in the special case where $O_{T'} = O_T$ and

c) in the special case where $L_{T'} \sqsubseteq_{O_T} L_T$.