463.9 Information Flow

CS463/ECE424

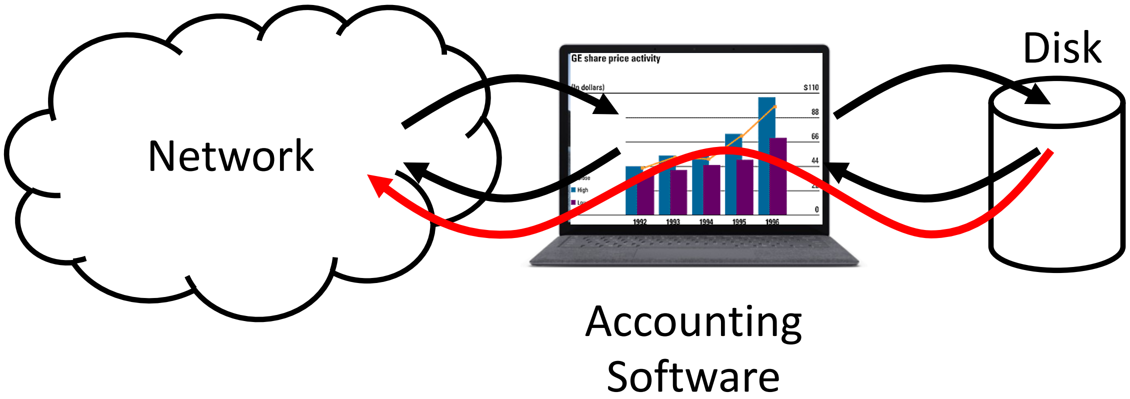
University of Illinois

Information Flow
Formal Model (two classic papers)

[GoguenM82J Security Policies and Security Models, J. A. Goguen and J. Meseguer. IEEE Security and Privacy 1982.
[DenningD77] Certification of Programs for Secure Information Flow, Dorothy E. Denning and Peter J. Denning. CACM 20(7), 1977.

Example: Financial Planner

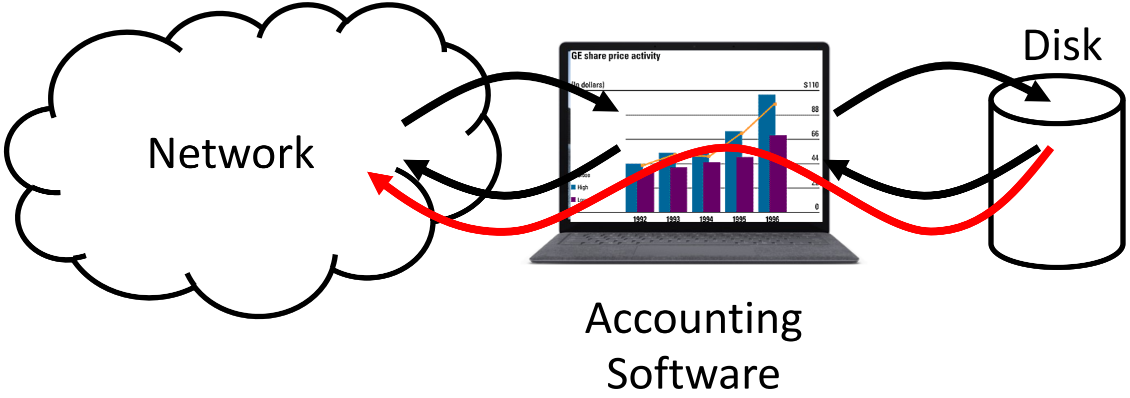
- Downloadable financial planner software:



  - Access control insufficient
  - Encryption necessary, but also insufficient

Noninterference

- Downloadable financial planner software:



  - Private data does not *interfere* with network communication
  - Baseline confidentiality policy

Model of Noninterference

- Represent noninterference as a relation between groups of users and commands

- Users in group G do not interfere with those in group G’ if the state seen by G’ is not affected by the commands executed by members of G

- Example: hotel rooms
  - Infer people’s activities based on side channels

[GoguenM82J Security Policies and Security Models, J. A. Goguen and J. Meseguer. IEEE Security and Privacy 1982.

State Automaton

- U – Users
- S – States
- C – Commands
- Out – Outputs
- do : $S \times U \times C \rightarrow S$ – state transition function
- out : $S \times U \rightarrow Out$ – output function
- s_0 – initial machine state

Capability System

- U, S, Out – users, states, commands, and outputs as before
- Capt – Capability tables (*defines permissions available to users*)
- SC – State commands
- CC – Capability commands
- out : $S \times Capt \times U \rightarrow Out$
- do : $S \times Capt \times U \times SC \rightarrow S$
- cdo : $Capt \times U \times CC \rightarrow Capt$ – Capability selection function
  - Give users a new permission or update the users' permissions
- $s_0 \in S$ and $t_0 \in Capt$ – Initial state and capability tables

Transition Function

- $C = SC \uplus CC$ - Commands
- $csdo : S \times Capt \times U \times C \rightarrow S \times Capt$
  - $csdo(s,t,u,c) = (do(s,t,u,c),t)$ if $c \in SC$
  - $csdo(s,t,u,c) = (s,cdo(s,t,u,c))$ if $c \in CC$
- $csdo^* : S \times Capt \times (U \times C)^* \rightarrow S \times Capt$
  - $csdo^*(s,t,nil) = (s,t)$
  - $csdo^*(s,t,w.(u,c)) = csdo(csdo^*(s,t,w),u,c)$
- $[[w]] = csdo^*(s_0,t_0,w)$
- $[[w]]_u = out([[w]],u)$

Chaining

Output the states visible to user u

Projection

- Let $G \subseteq U$ and $A \subseteq C$ and $w \in (U \times C)^*$
- $P_G(w)$ = subsequence of w obtained by eliminating pairs (u,c) where $u \in G$
- $P_A(w)$ = subsequence of w obtained by eliminating pairs (u,c) where $c \in A$
- $P_{G,A}(w)$ = subsequence of w obtained by eliminating pairs (u,c) where $u \in G$ and $c \in A$

Define Noninterference G :| G′

G does not interferer with G′

- M state machine and G, G’ $\subseteq$ U and A $\subseteq$ C

- G :| G’ iff $\forall$ w $\in$ (U $\times$ C)*. $\forall$ u $\in$ G’. $[[w]]_u = [[p_G(w)]]_u$

- A :| G iff $\forall$ w $\in$ (U $\times$ C)*. $\forall$ u $\in$ G. $[[w]]_u = [[p_A(w)]]_u$

- A,G :| G’ iff $\forall$ w $\in$ (U $\times$ C)*. $\forall$ u $\in$ G’. $[[w]]_u = [[p_{A,G}(w)]]_u$

Security Policies

- *Noninterference assertions* have the forms

  G :| G’

  A :| G

  A,G :| G’

- A *security policy* is a set of noninterference assertions

Example 1

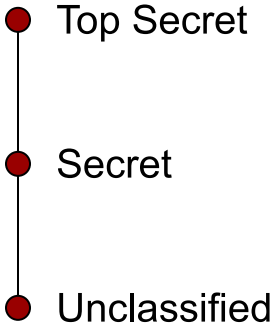
- A :| {u}
- The commands in A do not interfere with the state of user u

Example 2 Multilevel Security (MLS) and BLP Model

Less than or equal to

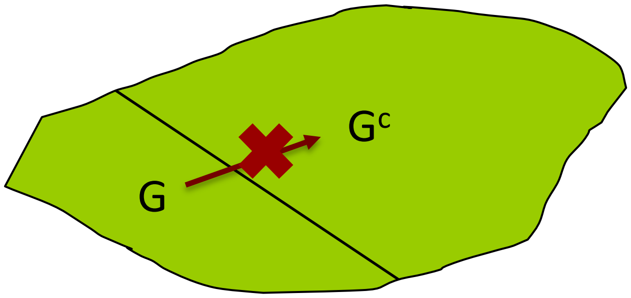
- Level : U → L
  - Assignment of security levels in L
- Above(λ) = { u ∈ U | λ ⊑ Level(u)}
- Below(λ) = { u ∈ U | Level(u) ⊑ λ}
- M is *multi-level secure* with respect to L if, for all λ ⊏ λ' in L, Above(λ') :| Below(λ)

Levels L ⊑



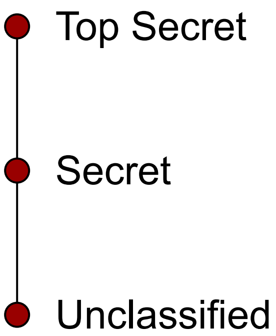
MLS Continued

- G is *invisible* if G :| G^c where G^c is the complement of G in U



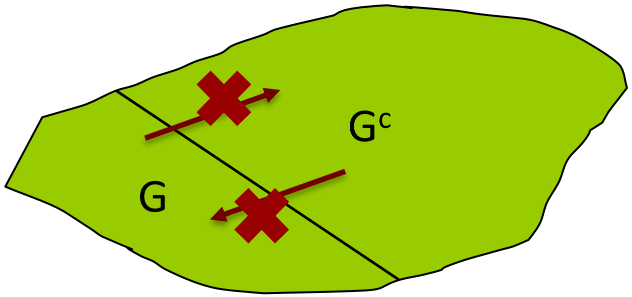
- **Proposition 1**: *If M,L is multi-level secure, then Above(λ) is invisible for every λ ∈ L.*

Levels L ⊑



Example 4 Isolation

- A group of users G is *isolated* if: G :| G^c and G^c :| G.
- A system is *completely* isolated if every user in U is isolated.



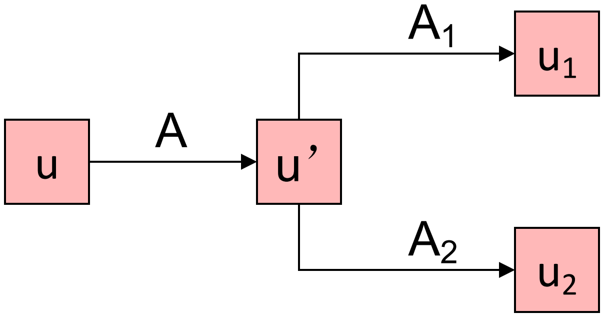
Example 5 Channel Control

- View a *channel* as a set of commands A
- We can assert that groups of users G and G’ can only communicate through channel A with the following two noninterference assertions:

  A^c,G :| G’

  A^c,G’ :| G

Example 6 Information Flow



$u', u_1, u_2 :| u$

$u_1, u_2 :| u'$

$u_1 :| u_2$

$u_2 :| u_1$

$A^c, u :| \{u', u_1, u_2\}$

$A_1^c, u' :| \{u_1\}$

$A_2^c, u' :| \{u_2\}$

Example 7 Security Officer

- Let A be the set of commands that can change the security policy
- seco ∈ U is the only individual permitted to use these commands to make changes
- This is expressed by the following policy: $A, \{seco\}^c :| U$

Entropy and Information Flow

- It is possible to analyze information flows in programs with an information theory foundation

- Intuition: info flows from *x* to *y* as a result of a sequence of commands *c* if you can deduce information about *x* before *c* from the value in *y* after *c*

$$x \xrightarrow{c} y$$

[DenningD77] Certification of Programs for Secure Information Flow, Dorothy E. Denning and Peter J. Denning. CACM 20(7), 1977. http://seclab.uiuc.edu/docs/DenningD77.pdf

Example 1

- y := x (*assign value x to variable y*)
  - If we learn y, then we know x
  - Clearly information flows from x to y

Example 2

- Suppose we are given

  r := x

  r := r - r

  y := 1 + r

- Does information flow from x to y?
- It does not, because r = 0 after the second command
  - There is no information flowing from x to y

Example 3

- Consider this branching command:

  if $x = 1$ **then** $y := 0$
  else $y := 1$;

- If we find after this command that y is 0, then we know that x was 1
- So information flowed from *x* to *y*

Implicit Flow of Information

- Information flows from x to y without an *explicit* assignment of the form $y := f(x)$ where $f(x)$ an arithmetic expression with variable x
- Recall the example from previous slide:

  if $x = 1$ **then** $y := 0$

  else $y := 1$;
- So we must look for *implicit* flows of information to analyze program

Conservative Automated Analysis of Flow

- Example 2 depends on an arithmetic property of subtraction
  - “r – r = 0”
- It is impossible to take each such property into account when doing an automated analysis
  - Ultimately undecidable
- Hence an automated analysis will be a conservative approximation of information flows
  - All flows can be found (even if trivially!)
  - Some non-flows (false positives) will be found

Compiler-Based Mechanisms

If a variable contains high-security information, does the information leak to low-security variables?

- Detect unauthorized information flows in a program during compilation
- Analysis not precise (may have false positives), but secure
  - If a flow *could* violate policy (but may not), it is unauthorized
  - No unauthorized path along which information could flow remains undetected
- Set of statements *certified* with respect to information flow policy if flows in set of statements do not violate that policy

Example

$\underline{x}$ is the security class of x

```
if x = 1 then y := a else y := b;
```

- Info flows from *x* and *a* to *y*, or from *x* and *b* to *y*
- Certified only if information from the security class $\underline{x}$ of x is allowed to flow into the security class $\underline{y}$ of y and similar conditions hold for a and b relative to y.
- Write: $\underline{x} \le \underline{y}$ and $\underline{a} \le \underline{y}$ and $\underline{b} \le \underline{y}$
  - Note flows for *both* branches must be true unless compiler can determine that one branch will *never* be taken

Declarations

“lub”: least upper bound

x: int class {A,B}

- Means x is an integer variable with security class at least lub{ A, B } so lub{ A, B } ≤ $\underline{x}$.
- Basic case is two security classes, High and Low.

Assignment Statements

```
x := y + z;
```

- Information flows from *y*, *z* to *x*
- this requires lub{$\underline{y}$, $\underline{z}$ } ≤ $\underline{x}$

More generally:

$$y := f(x_1, \ldots, x_n)$$

- Require lub{ $\underline{x}_1$, …, $\underline{x}_n$ } ≤ $\underline{y}$

Compound Statements

```
x := y + z;
a := b * c - x;
```

- First statement: lub{$\underline{y}$, $\underline{z}$ } ≤ $\underline{x}$
- Second statement: lub{$\underline{b}$, $\underline{c}$, $\underline{x}$ } ≤ $\underline{a}$
- So, both must hold (i.e., be secure)

More generally:

S_1; … S_n;

- Each individual S_i must be secure

Iterative Statements

```
while i < n do
begin a[i] := b[i]; i := i + 1; end
```

- Same ideas as for "if", but must terminate

More generally:

$\texttt{while f(}x_1\texttt{, …, }x_n\texttt{) do S;}$

"glb": greatest lower bound

- S must be secure
- lub{ $\underline{x}_1, \ldots, \underline{x}_n$ } ≤ glb{$\underline{y}$ | y target of an assignment in S }
- Loop must terminate

Conditional Statements

```
if x + y < z
then a := b
else d := b * c - x; end
```

- The statement executed reveals information about x, y, z, so lub{ $\underline{x}$, $\underline{y}$, $\underline{z}$ } ≤ glb{ $\underline{a}$, $\underline{d}$ }

More generally:

```
if f(x₁, …, xₙ) then S₁ else S₂; end
```

- S_1, S_2 must be secure
- lub{ $\underline{x}_1$, …, $\underline{x}_n$ } ≤ glb{$\underline{y}$ | y target of assignment in S_1, S_2 }

begin	
i, n: **integer security class** L;	
flag: **Boolean security class** L;	
$f1, f2$: **file security class** L;	
x, sum: **integer security class** H;	
$f3, f4$: **file security class** H;	
begin	
$i := 1$;	$\underline{1} \rightarrow \underline{i}\ (L \rightarrow L)$
$n := 0$;	$\underline{0} \rightarrow \underline{n}\ (L \rightarrow L)$
$sum := 0$;	$\underline{0} \rightarrow \underline{sum}\ (L \rightarrow H)$
while $i \leq 100$ **do**	
begin	
input *flag* **from** $f1$;	$\underline{f1} \rightarrow \underline{flag}\ (L \rightarrow L)$
output *flag* **to** $f2$;	$\underline{flag} \rightarrow \underline{f2}\ (L \rightarrow L)$
input x **from** $f3$;	$\underline{f3} \rightarrow \underline{x}\ (H \rightarrow H)$
if *flag* **then**	
begin	
$n := n + 1$;	$\underline{n} \oplus \underline{1} \rightarrow \underline{n}\ (L \rightarrow L)$
$sum := sum + x$	$\underline{sum} \oplus \underline{x} \rightarrow \underline{sum}\ (H \rightarrow H)$
end;	$\underline{flag} \rightarrow \underline{n} \otimes \underline{sum}\ (L \rightarrow L)$
$i := i + 1$	$\underline{i} \oplus \underline{1} \rightarrow \underline{i}\ (L \rightarrow L)$
end;	$\underline{i} \oplus \underline{100} \rightarrow \underline{flag} \otimes \underline{f2} \otimes \underline{x} \otimes \underline{n} \otimes \underline{sum} \otimes \underline{i}\ (L \rightarrow L)$
output $n, sum, sum/n$ **to** $f4$	$\underline{n} \oplus \underline{sum} \oplus \underline{sum} \oplus \underline{n} \rightarrow \underline{f4}\ (H \rightarrow H)$
end	
end	

Need to Handle More

- Procedures
- Arrays
- Goto Statements
- Exceptions
- Infinite loops
- Concurrency
- Etc

Reading

- [Bishop03] Computer Security Art and Science, Matt Bishop, Addison Wesley, 2003.
  - Chapter 8 up to the beginning of 8.2.1.
  - Chapter 16 sections 16.1 and 16.3
- [GoguenM82J Security Policies and Security Models, J. A. Goguen and J. Meseguer. IEEE Security and Privacy 1982.
- [DenningD77] Certification of Programs for Secure Information Flow, Dorothy E. Denning and Peter J. Denning. CACM 20(7), 1977.

Case Studies

Audit

Consider the security officer in example 7: seco ∈ U is the only individual permitted to use these commands to make changes

Shouldn’t the officer see audit information from the users who attempt to execute security commands?

Secret Communication

A general tells his army that if they see a green flag they should attack from the left but if they see a red flag they should attack from the right.

The general raises the green flag and the enemy forces see this.

Did the signal “interfere” with the enemy?