463.8 Information Flow

Computer Security II
CS463/ECE424
University of Illinois
A federal judge on Friday gave final approval to a $650 million Facebook class action privacy settlement and ordered the 1.6 million members of the class in Illinois who submitted claims to be paid “as expeditiously as possible.”

Chicago attorney Jay Edelstone sued Facebook in Cook County Circuit Court back in 2015, alleging that the platform’s use of facial recognition tagging was not allowed under the Illinois Biometric Information Privacy Act. The lawsuit claimed that Facebook's Tag Suggestions tool, which scanned faces in users’ photos and offered suggestions about who the person might be, stored biometric data without users' consent in violation of the Illinois law.
One More Thing

- Deep Nostalgia


Deep Nostalgia: 'creepy' new service uses AI to animate old family photos

Service from MyHeritage uses deep learning technique to automatically animate faces
Information Flow
Formal Model (two classic papers)

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
Example: Payment Card for Financial Transactions

- Card contains sensitive financial info used to make a transaction
- Used in the terminal of a merchant
- Card is tamper-resistant, protects card secrets from both the merchant and card holder
- Java cards provide limited card-based computation
- Must split the computation between the card, the terminal, and network connections
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

State Automaton

• U – Users
• S – States
• C – Commands
• Out – Outputs
• do : S × U × C → S – state transition function
• out : S × U → Out – output function
• s₀ – initial machine state
Capability System

• U, S, Out – users, states, commands, and outputs as before
• SC – State commands
• Capt – Capability tables (defines permissions available to users)
• CC – Capability commands
• out : S × Capt × U → Out
• do : S × Capt × U × SC → S
• cdo : Capt × U × CC → Capt – Capability selection function
  – Give users a new permission or update the users’ permissions
• s₀ ∈ S and t₀ ∈ Capt – Initial state and capability tables
Transition Function

- $C = SC \cup CC$ - 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 $\forall w \in (U \times C)^{\ast}, \forall u \in G'. [[[w]]_u = [[[p_G(w)]]_u}$

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

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

- Level : $U \rightarrow L$
  - Assignment of security levels in $L$
- $\text{Above}(\lambda) = \{ u \in U | \lambda \sqsubseteq \text{Level}(u) \}$
- $\text{Below}(\lambda) = \{ u \in U | \text{Level}(u) \sqsubseteq \lambda \}$
- $M$ is multi-level secure with respect to $L$ if, for all $\lambda \sqsubseteq \lambda'$ in $L$, $\text{Above}(\lambda') :|\text{Below}(\lambda)$
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 $Above(\lambda)$ is invisible for every $\lambda \in L$. 

Levels L ⊑

- Top Secret
- Secret
- Unclassified
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

\[ u', u_1, u_2 :\vdash u \]
\[ u_1, u_2 :\vdash u' \]
\[ u_1 :\vdash u_2 \]
\[ u_2 :\vdash u_1 \]
\[ A'_{\text{c}}, u :\vdash \{u', u_1, u_2\} \]
\[ A_{\text{c}}, u :\vdash \{u_1\} \]
\[ A_1'_{\text{c}}, u' :\vdash \{u_1\} \]
\[ A_2'_{\text{c}}, u' :\vdash \{u_2\} \]
Example 7 Security Officer

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

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:

\[
\text{if } x = 1 \text{ then } y := 0 \\
\text{else } y := 1;
\]

• 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

• 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

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

```plaintext
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 \( x \) of \( x \) is allowed to flow into the security class \( y \) of \( y \) and similar conditions hold for \( a \) and \( b \) relative to \( y \).
- Write: \( x \leq y \) and \( a \leq y \) and \( b \leq y \)
  - Note flows for both branches must be true unless compiler can determine that one branch will never be taken
x: int class \{A,B\}

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

“\text{lub}”: least upper bound
Assignment Statements

\( x := y + z; \)
- Information flows from \( y, z \) to \( x \)
- this requires \( \text{lub}\{y, z\} \leq x \)

More generally:

\( y := f(x_1, \ldots, x_n) \)
- Require \( \text{lub}\{x_1, \ldots, x_n\} \leq y \)
Compound Statements

\( x := y + z; \)
\( a := b * c - x; \)

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

More generally:
\( S_1; \ldots \ S_n; \)
- 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:
while $f(x_1, ..., x_n)$ do $S$;

• $S$ must be secure
• $\text{lub}\{x_1, ..., x_n\} \leq \text{glb}\{y | y \text{ target of an assignment in } S\}$
• Loop must terminate

“glb”: greatest lower bound
### Conditional Statements

```
if \( x + y < z \)
then \( a := b \)
else \( d := b \cdot c - x; \) end
```

• The statement executed reveals information about \( x, y, z \), so \( \text{lub}\{ x, y, z \} \leq \text{glb}\{ a, d \} \)

More generally:
```
if \( f(x_1, ..., x_n) \) then \( S_1 \) else \( S_2; \) end
```

• \( S_1, S_2 \) must be secure

• \( \text{lub}\{ x_1, ..., x_n \} \leq \text{glb}\{ y | y \text{ target of assignment in } S_1, S_2 \} \)
begin
  \( i, n: \) integer security class \( L \);
  \( \text{flag}: \) Boolean security class \( L \);
  \( f1, f2: \) file security class \( L \);
  \( x, \text{sum}: \) integer security class \( H \);
  \( f3, f4: \) file security class \( H \);
begin
  \( i := 1; \)
  \( n := 0; \)
  \( \text{sum} := 0; \)
  while \( i \leq 100 \) do
    begin
      \( \text{input} \text{flag} \text{from} f1; \)
      \( \text{output} \text{flag} \text{to} f2; \)
      \( \text{input} x \text{from} f3; \)
      if \text{flag} then
        begin
          \( n := n + 1; \)
          \( \text{sum} := \text{sum} + x \)
        end;
      \( i := i + 1 \)
    end;
  \( \text{output} n, \text{sum}, \text{sum}/n \text{to} f4 \)
end
Need to Handle More

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

  - Chapter 8 up to the beginning of 8.2.1.
  - Chapter 16 sections 16.1 and 16.3


Discussion

Is the G/M notion of interference too strong?

Consider the following two case studies.
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?