Course outline: the four hours

1. Language-Based Security: motivation
2. Language-Based Information-Flow Security: the big picture
3. Dimensions and principles of declassification
4. Dynamic vs. static security enforcement
Dimensions of Declassification in Theory and Practice
Confidentiality: preventing information leaks

- Untrusted/buggy code should not leak sensitive information
- But some applications depend on intended information leaks
  - password checking
  - information purchase
  - spreadsheet computation
  - ...
- Some leaks must be allowed: need information release (or declassification)
Confidentiality vs. intended leaks

- Allowing leaks might compromise confidentiality
- Noninterference is violated
- How do we know secrets are not laundered via release mechanisms?
- Need for security assurance for programs with release
State-of-the-art

- Related noninterference
- Admissibility
- Harmless flows
- Partial security
- Relative secrecy
- Selective flows
- Noninterference “until”
- Conditional noninterference
- Conditioned noninterference
- Robust declassification
- Intransitive noninterference
- Delimited release
- Abstract noninterference
- Computational security
- Approximate noninterference
- Constrained noninterference
- Relaxed noninterference
Principles of release

1. Semantic consistency
2. Conservativity
3. Monotonicity
4. Non-occlusion
What

• Noninterference [Goguen & Meseguer]: as high input varied, low-level outputs unchanged

h₁ → h₁' → h₂' → h₂

| | | | |

• Selective (partial) flow
  – Noninterference within high sub-domains [Cohen’78, Joshi & Leino’00]
  – Equivalence-relations view [Sabelfeld & Sands’01]
  – Abstract noninterference [Giacobazzi & Mastroeni’04,’05]
  – Delimited release [Sabelfeld & Myers’04]

• Quantitative information flow [Denning’82, Clark et al.’02, Lowe’02]
Security lattice and noninterference

Noninterference: flow from $l$ to $l'$ allowed when $l \subseteq l'$
Noninterference

- Noninterference [Goguen & Meseguer]: as high input varied, low-level outputs unchanged

\[ h_1 \rightarrow h_1' \quad h_2 \rightarrow h_2' \]

- Language-based noninterference for \( c \):

\[ M_1 =_L M_2 \land \langle M_1, c \rangle \downarrow M_1' \land \langle M_2, c \rangle \downarrow M_2' \Rightarrow M_1' =_L M_2' \]

Low-memory equality: \( M_1 =_L M_2 \) iff \( M_1|_L = M_2|_L \)

Configuration with \( M_2 \) and \( c \)
Average salary

- Intention: release average
  
  \[
  \text{avg} := \text{declassify}((h_1 + \ldots + h_n)/n, \text{low});
  \]

- Flatly rejected by noninterference
- If accepting, how do we know declassify does not release more than intended?
- Essence of the problem: what is released?
- “Only declassified data and no further information”
- Expressions under declassify: “escape hatches”
Delimited release
[Sabelfeld & Myers, ISSS’03]

• Command $c$ has expressions $\text{declassify}(e_i, L)$; $c$ is secure if:

$$
M_1 =_L M_2 \land \langle M_1, c \rangle \downarrow M'_1 \land \langle M_2, c \rangle \downarrow M'_2 \land
\forall i \cdot \text{eval}(M_1, e_i) = \text{eval}(M_2, e_i) \Rightarrow
M'_1 =_L M'_2
$$

$\Rightarrow$ security

• For programs with no declassification:
  Security $\Rightarrow$ noninterference

if $M_1$ and $M_2$ are indistinguishable through all $e_i$...

...then the entire program may not distinguish $M_1$ and $M_2$
Average salary revisited

• Accepted by delimited release:

\[
\text{avg} := \text{declassify}\left(\frac{h_1 + \ldots + h_n}{n}, \text{low}\right);
\]

\[
\text{temp} := h_1; h_1 := h_2; h_2 := \text{temp};
\]

\[
\text{avg} := \text{declassify}\left(\frac{h_1 + \ldots + h_n}{n}, \text{low}\right);
\]

• Laundering attack rejected:

\[
\begin{align*}
h_2 &:= h_1; \ldots; h_n := h_1; \\
\text{avg} &:= \text{declassify}\left(\frac{h_1 + \ldots + h_n}{n}, \text{low}\right);
\end{align*}
\]

\[
\sim \quad \text{avg} := h_1
\]
Electronic wallet

• If enough money then purchase

\[
\text{if declassify}(h \geq k, \text{low}) \text{ then } (h := h - k; l := l + k);
\]

• Accepted by delimited release

amount in wallet  cost  spent
Electronic wallet attack

• Laundering bit-by-bit attack ($h$ is an $n$-bit integer)

```plaintext
l := 0;
while (n ≥ 0) do
    k := 2^{n-1};
    if declassify($h ≥ k, low$) then
        (h := h - k; l := l + k);
    n := n - 1;
```

• Rejected by delimited release

\[ l := h \]
Security type system

- Basic idea: prevent new information from flowing into variables used in escape hatch expressions

  - Theorem:
    
    \[ c \text{ is typable} \implies c \text{ is secure} \]

\[
\begin{align*}
  h & := \ldots; \\
  \ldots & \\
  \text{declassify}(h, \text{low}) & \\
\end{align*}
\]

\[
\begin{align*}
  \text{while } \ldots \text{ do} \\
  \text{declassify}(h, \text{low}) & \\
  \ldots & \\
  h & := \ldots; \\
\end{align*}
\]

- May not use other (than \( h \)) high variables

\[
\begin{align*}
  \text{may not use} \\
  \text{other (than } h \text{)} \text{ high variables} & \\
\end{align*}
\]
Who

- Robust declassification in a language setting [Myers, Sabelfeld & Zdancewic’04/06]
- Command $c[\bullet]$ has robustness if

$$\forall M_1, M_2, a, a'. \langle M_1, c[a] \rangle \approx_L \langle M_2, c[a] \rangle \Rightarrow \langle M_1, c[a'] \rangle \approx_L \langle M_2, c[a'] \rangle$$

- If $a$ cannot distinguish bet. $M_1$ and $M_2$ through $c$ then no other $a'$ can distinguish bet. $M_1$ and $M_2$
Robust declassification: examples

- Flatly rejected by noninterference, but secure programs satisfy robustness:

  \[
  \text{[\bullet]; } x_{LH} := \text{declassify}(y_{HH}, LH) \\
  \text{[\bullet]; if } x_{LH} \text{ then } y_{LH} := \text{declassify}(z_{HH}, LH)
  \]

- Insecure program:

  \[
  \text{[\bullet]; if } x_{LL} \text{ then } y_{LL} := \text{declassify}(z_{HH}, LH)
  \]

is rejected by robustness
Enforcing robustness

- Security typing for declassification:

\[ \text{LH} \vdash e : \text{HH} \]

\[ \text{LH} \vdash \text{declassify}(e, l') : \text{LH} \]

context must be high-integrity

data must be high-integrity
Where

- Intransitive (non)interference
  - assurance for intransitive flow [Rushby’92, Pinsky’95, Roscoe & Goldsmith’99]
  - nondeterministic systems [Mantel’01]
  - concurrent systems [Mantel & Sands’04]
  - to be declassified data must pass a downgrader [Ryan & Schneider’99, Mullins’00, Dam & Giambiagi’00, Bossi et al.’04, Echahed & Prost’05, Almeida Matos & Boudol’05]
When

- **Time-complexity based attacker**
  - password matching [Volpano & Smith’00] and one-way functions [Volpano’00]
  - poly-time process calculi [Lincoln et al.’98, Mitchell’01]
  - impact on encryption [Laud’01,’03]

- **Probabilistic attacker** [DiPierro et al.’02, Backes & Pfitzmann’03]

- **Relative: specification-bound attacker** [Dam & Giambiagi’00,’03]

- **Non-interference “until”** [Chong & Myers’04]
The (in)security of a program is invariant under semantics-preserving transformations of declassification-free subprograms.

- Aid in modular design
- “What” definitions generally semantically consistent
- Uncovers semantic anomalies
Principle II

• Straightforward to enforce (by definition); nevertheless:
• Noninterference “until” rejects

if h > h then l := 0
Principle III

Monotonicity of release

Adding further declassifications to a secure program cannot render it insecure.

- Or, equivalently, an insecure program cannot be made secure by removing declassification annotations.

- "Where": intransitive noninterference (a la M&S) fails it; declassification actions are observable.

```
if h then declassify(l=l) else l=l
```
Principle IV

Occlusion

The presence of a declassification operation cannot mask other covert declassifications
Checking the principles

<table>
<thead>
<tr>
<th>What</th>
<th>Property</th>
<th>Semantic consistency</th>
<th>Conservativity</th>
<th>Monotonicity of release</th>
<th>Non-occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Partial release [Coh78, JL00, SS01, GM04, GM05]</td>
<td>✓</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Delimited release [SM04]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Relaxed noninterference [LZ05a]</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Naive release</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Who</th>
<th>Property</th>
<th>Semantic consistency</th>
<th>Conservativity</th>
<th>Monotonicity of release</th>
<th>Non-occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Robust declassification [MSZ04]</td>
<td>✓*</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Qualified robust declassification [MSZ04]</td>
<td>✓*</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Where</th>
<th>Property</th>
<th>Semantic consistency</th>
<th>Conservativity</th>
<th>Monotonicity of release</th>
<th>Non-occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intransitive noninterference [MS04]</td>
<td>✓*</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>When</th>
<th>Property</th>
<th>Semantic consistency</th>
<th>Conservativity</th>
<th>Monotonicity of release</th>
<th>Non-occlusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Admissibility [DG00, GD03]</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Noninterference “until” [CM04]</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Typeless noninterference “until”</td>
<td>✓*</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

* Semantic anomalies
Declassification in practice: A case study

[Askarov & Sabelfeld, ESORICS’05]

- Use of security-typed languages for implementation of crypto protocols
- Mental Poker protocol by [Roca et.al, 2003]
  - Environment of mutual distrust
  - Efficient
- Jif language [Myers et al., 1999-2005]
  - Java extension with security types
  - Decentralized Label Model
  - Support for declassification
- Largest code written in security-typed language up to publ date [~4500 LOC]
## Security assurance/Declassification

<table>
<thead>
<tr>
<th>Group</th>
<th>Pt.</th>
<th>What</th>
<th>Who</th>
<th>Where</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
<td>Public key for signature</td>
<td>Anyone</td>
<td>Initialization</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>Public security parameter</td>
<td>Player</td>
<td>Initialization</td>
</tr>
<tr>
<td>II</td>
<td>3</td>
<td>Message signature</td>
<td>Player</td>
<td>Sending msg</td>
</tr>
<tr>
<td>II</td>
<td>4-7</td>
<td>Protocol initialization data</td>
<td>Player</td>
<td>Initialization</td>
</tr>
<tr>
<td>II</td>
<td>8-10</td>
<td>Encrypted permuted card</td>
<td>Player</td>
<td>Card drawing</td>
</tr>
<tr>
<td>III</td>
<td>11</td>
<td>Decryption flag</td>
<td>Player</td>
<td>Card drawing</td>
</tr>
<tr>
<td>IV</td>
<td>12-13</td>
<td>Player’s secret encryption key</td>
<td>Player</td>
<td>Verification</td>
</tr>
<tr>
<td>IV</td>
<td>14</td>
<td>Player’s secret permutation</td>
<td>Player</td>
<td>Verification</td>
</tr>
</tbody>
</table>

- **Group I** – naturally public data
- **Group II** – required by crypto protocol
- **Group III** – success flag pattern
- **Group IV** – revealing keys for verification
Dimensions: Conclusion

- **Road map** of information release in programs
- Step towards **policy perimeter defense**: to protect along each dimension
- Prudent **principles** of declassification (uncovering previously unnoticed anomalies)
- Need for declassification framework for relation and combination along the dimensions
References

• Declassification: Dimensions and Principles
  [Sabelfeld & Sands, JCS]