Language-Based Information-Flow Security
Andrei Sabelfeld  Andrew C. Myers
Cornell University

survey appeared in
IEEE-JSAC, Jan. 2003
Dagstuhl
Oct. 2003

A scenario: free service software
Users freely download and use the software providing a service:
- Grokster, Kazaa, Morpheus,... are file-sharing services helping users exchange files
- Come with ”hooks” for automatic updates
- Support advertisement to justify cost

Real story: malware
Users are tricked to download software bundled with:
- Homepage/search hijackers (MySearch)
- Unsolicited pop-up ads
- Rewriting URLs to override original ads with own
- ”Hooks” for automatic updates are used to execute the advertiser’s arbitrary code (MediaUpdate, DownLoadware)
- Information gathering—visited URLs and filled forms are forwarded to a third-party (Gator, IPInsight, Transponder)

General problem: malicious and/or buggy code is a threat
- Trends in software
  - mobile code, executable content
  - platform-independence
  - extensibility
- These trends are attackers’ opportunities!
  - easy to distribute worms, viruses, exploits,...
  - write (an attack) once, run everywhere
  - systems are vulnerable to undesirable modifications
- Need to keep the trends without compromising information security

Language-based security
- Looking under the street light...
  Attacker model:
  - eavesdropping on network
  - modifying network traffic
  - trusted communication endpoints
  ⇒ cryptographic protection of communication
- ...for a key that lies somewhere else!
  Real story [CERT]: Most attacks are
  - remote penetrations (buffer overruns, format strings, RPC vulnerabilities,...)
  - malware (viruses, worms, DDoS slaves,...)
  ⇒ need protection at application level

Information security: confidentiality
- Confidentiality: sensitive information must not be leaked by computation (non-example: spyware attacks)
- End-to-end confidentiality: there is no insecure information flow through the system
- Standard security mechanisms provide no end-to-end guarantees
  - Security policies too low-level (legacy of OS-based security mechanisms)
  - Programs treated as black boxes
Confidentiality: standard security mechanisms

Access control
+ prevents “unauthorized” release of information
- but what process should be authorized?

Firewalls
+ permit selected communication
- permitted communication might be harmful

Encryption
+ secures a communication channel
- even if properly used, endpoints of communication may leak data

Confidentiality: language-based approach

- Counter application-level attacks at the level of a programming language—look inside the black box! Immediate benefits:
- Semantics-based security specification
  – End-to-end security policies
  – Powerful techniques for reasoning about semantics
- Static security analysis
  – Analysis enforcing end-to-end security
  – Track information flow via security types
  – Type checking by the compiler removes run-time overhead

Dynamic security enforcement

Java’s sandbox, OS-based monitoring, and Mandatory Access Control dynamically enforce security policies; But:

Problem: monitoring a single execution path is not enough!

Static certification

- Only run programs which can be statically verified as secure before running them
- Static certification for inclusion in a compiler [Denning & Denning’77]
- More precise implicit flow analysis
- Enforcement by static analysis (e.g., security-type systems)

A security-type system

Expressions:

\[ \text{exp} : \text{high} \quad \text{if} \quad h \notin \text{Vars} (\text{exp}) \quad \text{exp} : \text{low} \]

Atomic commands (pc represents context):

\[ [\text{pc}] \vdash \text{skip} \]

\[ [\text{pc}] \vdash h := \text{exp} \]

\[ [\text{pc}] \vdash \text{exp} : \text{low} \]

\[ [\text{low}] \vdash l := \text{exp} \]

\[ \text{context} \]
A security-type system: Compositional rules

Table:

<table>
<thead>
<tr>
<th>[high] ⊢ C</th>
<th>[pc] ⊢ C₁</th>
<th>[pc] ⊢ C₂</th>
</tr>
</thead>
</table>

Implicit flows: branches of a high if must be typable in a high context

Semantics-based security

- What end-to-end policy such a type system guarantees (if any)?
- Semantics-based specification of information-flow security [Cohen’77], generally known as noninterference [Goguen & Meseguer’82]:

A program is secure iff high inputs do not interfere with low-level view of the system

Semantics-based security

- What is ≈ for our language?
- Intention: [pc] ⊢ C ⇒ C is secure
  I.e., if C is typable then

∀S₁, S₂. S₁ =ₚ S₂
⇒ [ C ]S₁ ≈ₚ [ C ]S₂
⇔ [ C ]S₁ ⊢ₚ [ C ]S₂ ⇒ [ C ]S₁ =ₚ [ C ]S₂

Termination-insensitive interpretation of ≈ₚ

Evolution of language-based information flow

Before mid nineties two separate lines of work:
Static certification, e.g., [Denning & Denning’76, Bergeretti & Carre’85, Mizuno & Oldehoeft’87, Palser & Ørbaek’95]

Security specification, e.g., [Cohen’77, Andrews & Reitman’80, Banatre & Bryce’93, McLean’94]

Volpano et al.’96: First connection between noninterference and static certification: security-type system that enforces noninterference
Evolution of language-based information flow

Four main categories of current information-flow security research:

- Enriching language expressiveness
- Exploring impact of concurrency
- Analyzing covert channels (mechanisms not intended for information transfer)
- Refining security policies

Concurrence: Nondeterminism

- Possibilistic security: variation of \( h \) should not affect the set of possible \( l \)
- An elegant equational security characterization [Leino & Joshi'00]:
  suppose \( HH \) ("havoc on \( h\)") sets \( h \) to an arbitrary value; \( C \) is secure iff

\[
\forall s. [HH; C; HH] s \approx [C; HH] s
\]

Concurrence: Multi-threading

- The high data must be protected at all times: \( h := 0; \hat{l} := h \) is secure as a sequential program, but not when \( h := h' \) is run in parallel
- A type system [Smith & Volpano'98] for nondeterministically scheduled threads rejects high while loops, but not leaks via schedulers:

[Diagram showing scheduling and thread interactions]

- Encoding of a timing leak to a direct leak

Concurrence: Multi-threading

- A later work [Volpano & Smith'98] proposes a "protect" command for wrapping high if
  - Scheduler-independent security: no need for "protect" via Agat's transformation [Sabelfeld & Sands'00]
  - Thread synchronization (as by semaphores) may lead to leaks by blocking [Sabelfeld'01]
  - Permissive type systems for multithreaded programs [Boudal & Castelani'01, 02]
  - A uniform type system [Honda et al.'00, '02] and a light type system [Potter'02] for noninterference in \( \pi \)-calculus
  - Security through low determinism [Zdancewic & Myers'03]
Confidentiality issues for distributed systems

- Blocking of a process observable by other processes (also timing, probabilities, ...)
- Messages travel over publicly observable medium; encryption protects messages’ contents but not their presence
- Mutual distrust of components
- Components (hosts) may be compromised/subverted; messages may be delayed/lost

Concurrency: Distribution

- Jif/split: An architecture for secure program splitting to run on heterogeneously trusted hosts [Dancewic et al. ’01]
- Type systems for secrecy for cryptographic protocols in spi-calculus [Abadi’97, Abadi & Blanchet’01]
- Logical relations for the low view [Sumii & Pierce’01]
- Interplay between communication primitives and types of channels [Sabelfeld & Manter’02]
- Secure replication and partitioning [Zheng et al.’03]

Covert channels: Termination

- Covert channels are mechanisms not intended for information transfer
  
  Is while h>0 do h:=h+1 secure?

- Low view \( s \equiv_L s' \) iff \( s = s' \lor (s 
eq s' \land s =_L s') \)
- PER model can be naturally lifted to handle termination

Covert channels: Timing

- Nontermination \( \equiv_L \) time-consuming computation
- Bisimulation-based \( \equiv_L \) accurately expresses the observational power [Sabelfeld & Sands’00, Smith’01,’03]
- Agat’s cross-copying technique for transforming out timing leaks [Agat’00]

Covert channels: Probabilistic

- Possibilistically but not probabilistically secure:

  ```
  if h then sleep(100); l:=1
  else sleep(50); l:=0
  ```

- Probability-sensitive \( \equiv_L \) by PERs [Sabelfeld & Sands’99]
- Probabilistic bisimulation-based security [Volpano & Smith’99, Sabelfeld & Sands’00, Smith’01,’03]
Security policies

- Many programs intentionally release information, or perform declassification
- Noninterference is restrictive for declassification
  - Encryption
  - Password checking
  - Spreadsheet computation (e.g., tax preparation)
  - Database query (e.g., average salary)
  - Information purchase
- Most approaches to information flow control ignore declassification—need more flexible security policies

Security policies: Declassification

- To legitimize declassification we could add to the type system:

  \[
  \text{declassify}(h) : \text{low}
  \]

- But this violates noninterference
- What’s the right typing rule? What’s the security condition that allows intended declassifications?

Language-based information security: challenges

Some essential challenges—some are not addressed by current trends!

- System-wide security
- Certifying compilation
- Attacks beyond abstraction
- Dynamic policies
- Practical issues

  ⇒ Opportunities for integrating model checking, logic, theorem proving, code rewriting...

Conclusion

- Security practices not capable of tracking information flow
- Language-based security: effective information flow security models (semantics-based security) and enforcement mechanisms (security-type systems)
- Progress on expressive languages, concurrency, covert channels, security policies
- Critical challenges remain for language-based mechanisms to become a part of security practice
End of talk