Secure information flow: opportunities and challenges for security & forensics

Michael Huth¹

¹Department of Computing
Imperial College London

Talk for ACSF1, 13 July 2006
Liverpool, United Kingdom
Outline

1. Introduction
2. Security: opportunities
4. Computer forensics
Secure information flow, informal definition

“Information is stored, transformed, and flows between devices and agents. Such flow is considered **SECURE** if it abides by a specified policy, saying which information should be accessible when, where, and to whom.”

Are (types of) devices and agents known beforehand? Is policy adaptable, efficiently enforceable, local, comprehensible? Is information flow *either* secure *or* insecure? Are there useful metrics to assess inherent risks of such flow? Etc.
End-to-end security: motivation

- Access control systems (even a firewall) restrict access to information but, once granted access, *can’t control information propagation*.

- End-to-end security fundamental problem; e.g. recent trade secret case with Coca-Cola Enterprises Inc.

- This talk emphasizes end-to-end information flow within well specified system boundary.

- Example:
  - employees have restricted access within building, screened on entry and exit points (closed system)
  - employees aren’t monitored outside of company premises (outside of closed system).
Humans and artificial agents alike engage with intranets & internet through interfaces for computer programs:

- so systems of interacting computer programs our focus for secure information flow
- seek methods for specifying & verifying policies for secure information flow of computer programs.

Opportunities created:

- verification of secure flow expressible as a checkable certificate
- annotate code with its certificate
- efficient check of certificate establishes trust in code important for mobile code.
Example violations of secure information flow

```c
int l; // low-level security variable: PUBLIC
int h; // high-level security variable: SECRET

// direct violation (assignment)
l = h;
// entire information in h is being spilled

// indirect violation (control flow)
if ((h % 2) == 0) { l = 0; } else { l = 1; }
// one bit of information in h is being spilled

// subtle violation (termination behavior)
l = h; while (l != 0) { l = l * l; }
// learn whether h equals 0
```
Formal definition of secure information flow

- For each example program on previous slide, we “saw” that it violates secure information flow.
- Want checkable definition of secure information flow that correctly classifies these examples as insecure.
- End-to-end security catered for by such a definition based on input-output behavior: security thus focuses on “interface” boundary.
- Challenge: dependency on input-output behavior requires supporting compositionality principles for methods, classes, interfaces, packages and other modules.
Secure information as **non-interference**

“*Variation of confidential (high) input doesn’t cause variation of public (low) output.*”

[due to late J. A. Goguen & to J. Meseguer]

- **Low-level view:** \( s =_L s' \) means state \( s \) equals state \( s' \) modulo variations in any confidential (high) variables.

- **E.g.** \([l = -5, h = 1] =_L [l = -5, h = 3]\) as these states have same low-level view if \( l \) is low level, \( h \) high level.

- In particular, low-level view cannot directly observe any high-level inputs.
Non-interference and the attacker

- Attacker attempts to infer information about high-level inputs or outputs.
- **Limitations of attacker**: \( s \cong_L s' \), relation between states that specifies which states \( s \) and \( s' \) he or she can't distinguish
- \( \cong_L \) **configurable**, different notion for termination, timing, probabilities etc; e.g. \( \cong_L \) could be \( =_L \)
- **Formal non-interference**: For all states \( s \) and \( s' \),
  \[
  s =_L s' \Rightarrow \text{output}(\text{run } P \text{ at } s)) \cong_L \text{output}(\text{run } P \text{ at } s').
  \]
- Output of running \( P \) at \( s \) either genuine (e.g. “42”) or abstract (e.g. \( \perp \), expressing non-termination).
Approximate non-interference through types

- core programming language:

\[ C ::= \text{skip} \mid \text{var} = \text{exp} \mid C ; C \mid \]
\[ \text{if } \text{exp} \text{ then } C \text{ else } C \mid \text{while } \text{exp} \text{ do } C \]

- \( \tau \) element of finite security lattice \( L \)
  in this talk: \( L = \{ \text{low} < \text{high} \} \)

- security context \( \Gamma ::= \text{empty} \mid [\tau] \)

- two judgments, one for each clause of \( \Gamma \):
  - \([\tau] \vdash C\) expression \( C \) is typeable in security context \( \tau \)
  - \(\text{empty} \vdash C : \tau\) expression \( C \) has security type \( \tau \)
Type inference for non-interference

- inference rules for judgments, syntax-directed in $C$, e.g.
  - if no variable in $exp$ is high, then $\text{empty} \vdash exp : \text{low}$
  - if empty $\vdash exp : \text{low}$ and empty $\vdash var : \text{low}$ then
    $[\text{low}] \vdash var = exp : \text{low}$
  - if empty $\vdash exp : \tau$ and $[\tau] \vdash C$, then $[\tau] \vdash \text{while} exp \text{ do } C$
  - if empty $\vdash exp : \tau$, $[\tau] \vdash C_1$, and $[\tau] \vdash C_2$, then
    $[\tau] \vdash \text{if } exp \text{ then } C_1 \text{ else } C_2$
  - plus sub-sumption rule: “if $[\text{high}] \vdash C$, then $[\text{low}] \vdash C$”
    program typeable in high context, also typeable in low context
  - for empty $\vdash l : \text{low}$ and empty $\vdash h : \text{high}$
    - can derive $[\text{low}] \vdash h = l + 4; l = l - 5$
    - can derive $[\text{high}] \vdash \text{if } h == 1 \text{ then } h = h + 4 \text{ else skip}$
    - can’t derive $[\tau] \vdash \text{if } h == 1 \text{ then } l = 1$ for
      $\tau \in \{\text{low}, \text{high}\}$
Type inference: sound & trust-enabling security

- **Soundness**: if we derive $[\tau] \vdash C$ for some $\tau$, then for all $s =_L s'$: $output(\text{run } C \text{ at } s) =_L output(\text{run } C \text{ at } s')$

- So *typeable programs satisfy desired non-interference property*.

- **Opportunities**: proof-carrying code [Lee & Necula]
  - Type inference here very efficient. A bit more involved, but still efficient, for full-blown languages.
  - Program $C$ can be annotated with “proof” of type judgment $[\tau] \vdash C$, a complete type inference of that judgment.
  - Clients of $C$ may use very efficient type checking to validate that proof, building up trust into $C$ before running it.
Types: abstract but precise & scalable security

- Type inference & type checking are **compositional**: program secure if all subprograms are.
- Not all secure programs certifiable in this manner. E.g.
  - \( l = h + 1 \); \( l = 0 \) is non-interfering, end-to-end secure
  - its second subprogram \( l = 0 \) is secure
  - its first subprogram \( l = h + 1 \) is not; so for no \( \tau \) do we have
    \[
    [\tau] \vdash l = h + 1; l = 0
    \]
- Familiar **tradeoff** between precision (non-compositional) and efficiency (compositional) of analysis.
Some research issues

- work already done on extending this scalable type system to feature sets of modern languages such as C, Java, C#, e.g. multi-threading, exceptions, aspects ("harmless advice" [Dantas & Walker]), and remote procedure calls
- computer programs & systems subject to dynamically changing policy on secure information flow e.g. de-classifications & corruption of data
- hard to keep up with pace of languages’ development and their dissemination
- integration of proof-carrying code paradigm for secure information flow into software development tools.
Covert channels

- concurrency: information flow secure only relative to specified set of schedulers, e.g.:
  - if \( h = 1 \) then \( C_{long} \) else skip; \( l = 1 \) || \( l = 0 \) secure in "possibilistic" sense
  - if \( h \) equals 1 and \( C_{long} \) time-consuming command, then \( l = 1 \) likely to be last executed assignment
  - so timing leak can turn into direct leak
  - repeated execution may provide covert channel if attacker can control execution time of \( C_{long} \)

- our notion of non-interference did not cater for timing information and may be unsound for program above

- problem: soundness relative to fixed observability horizon which may have to be extended to time & probability
Quantitative information flow

Qualitative view “either program secure or it isn’t” tenable?

- entering two digits of your six-digit online-banking password may create quantitative information flow
- seek quantitative measure of such flow to assess security risk, e.g. for specific attacker context [Di Pierro, Hankin & Wiklicky]
- measure based on entropy & (conditional) mutual information [Shannon] suitable candidate since non-interference, as defined earlier, now captured as “measure equals 0” [Clark, Hunt & Malacaria]
- lower and upper bounds on such measures efficiently computable
- such safe but abstract bounds replace type system presented earlier [Clark et al.]; proof-carrying code?
Computer forensics, a definition

“Computer forensics is the collection, preservation, analysis and court presentation of computer-related evidence.”

[Barbin & Patzakis]

- contexts of use: e.g. theft of trade secrets or intellectual property, harassment claims, employment tribunals, arbitration, administrative proceedings, government hearings, presentations to upper management
- traditionally, used after alleged events happened
- now, preventive & documentary role grows: document in anticipation of bad events
Non-interference: collecting & preserving evidence

“... collection and preservation ... of computer-related evidence”

- Physics of observability, *can’t observe a system without interacting with it*:
  - If computer is ON, turn it OFF?
  - How to turn it OFF to minimize interference?
- State and programs of secured evidence (e.g. read-only hard drive) is high level *but may be known to low level forensic tools/experts*.
- Preservation process *must not interfere with* such high level state and its capabilities.
- **Non-interference** of preservation process may have to be provable in courts or for arbitration bodies.
(Non-)interference in analyzing evidence

“... analysis ... of computer-related evidence”

- Need to establish interference.
  - E.g. causal link between files on memory stick and files on disk drive; or recovery of deleted files.

- Need to establish non-interference.
  - E.g. forensic tools/experts need to demonstrate that HR Director’s “investigative” access of employee’s computer account didn’t interfere with evidence of employee’s actions on that account.

- Forensic analysis typically quantitative, based on risk & probabilities. E.g. other user may have cracked password.
(Non-)interference and presentation of evidence

“... and court presentation of computer-related evidence. ”

- Admissibility of evidence: accessed and preserved without breaking any laws in territories that host part of evidence?
- How conclusive is preserved evidence and its analysis?
- In particular, conclusive arguments need to demonstrate presence and absence of information flow at the right places:
  - e.g. gathered evidence not corrupted: absence of flow
  - e.g. deleted password file matches passwords posted on the internet: presence of flow
Computer forensics software

- **Preservation and analysis** of computer-related evidence largely *semi-automated*.
- **Automation** due to sophisticated computer *forensics software*.
- **Software reduces time and cost** of investigation, enables cost-effective preventive measures.
- **My naive questions to audience:**
  - “Can such software benefit from *formal but lightweight modeling and analysis* of computer systems’ capabilities and their inherent information flow?”
  - “Or is this *analysis best left to human forensic experts* who interpret the findings of existing tools?”
  - “Can such software be *verified*, and what would verification mean in the context of forensics?”
Prevention: **securing information flow**

- Aforementioned: computer forensics have growing preventive role.
- Desire to log significant events without excessive burden on working systems.
- Such logs hope to secure that relevant information can flow into tools for preservation and analysis.
- Example: modification of system kernel to map process origin in system process table [Buchholz & Shields]. Addresses network trace-back problem.
- Example: Deterministic replay of multi-threaded financial trading software. Facilitates audit of what may have gone wrong [Itskova].

Prevention: an **emerging area** in systems engineering and software engineering?
Conclusions

- Knowledge and technology transfer from formal aspects to mainstream slow but steady. E.g.
  - Took about 20 years for “model checking” to be embraced by Microsoft Inc. for verifying their driver software.
  - Advanced type systems now entered mainstream programming (such as session types for communication protocols).
  - Tools for verifying security protocol specifications now used routinely.

- Formal approaches to information security may be of interest to computer forensics community.
- Conversely, interests and needs of that community should inform research on formal aspects of computing.
- This conference is hoped to foster such interaction.
Acknowledgments

- Dr John Haggerty for organizational help
- Prof Madjid Merabti for having invited me