UNIVERSITY
OF OSLO

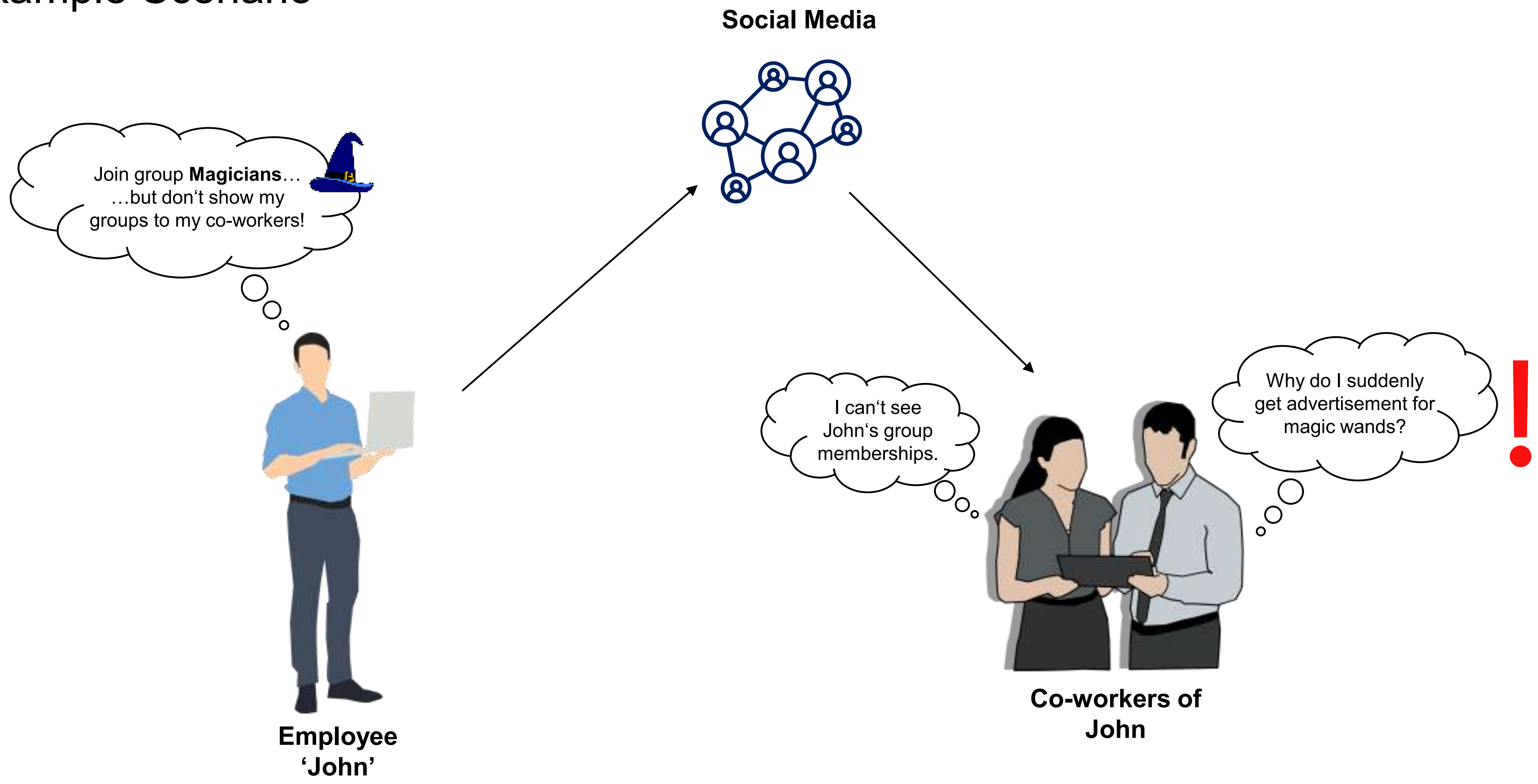
Trial Lecture:

Information Flow Properties for Security Policies on Data Usage

Andre Büttner

2nd September 2024

Example Scenario*



*Inspired by Kozyri et al. (2022)

Motivation

*Most work in computer security **has ignored the social contexts** in which systems are actually used*.

- Goguen and Meseguer (1982)

- Systems need to ensure safety of their users and their data
- Regulations like GDPR enforce measures to protect user data
- Complexity of systems makes verification of and adhering to security policies difficult
- Tracing information flow to detect policy violations

What is information?

- Directly shared data
- Metadata (sender, receiver, time, …)
- Network traffic
- System behavior
- And more…

Security Policies

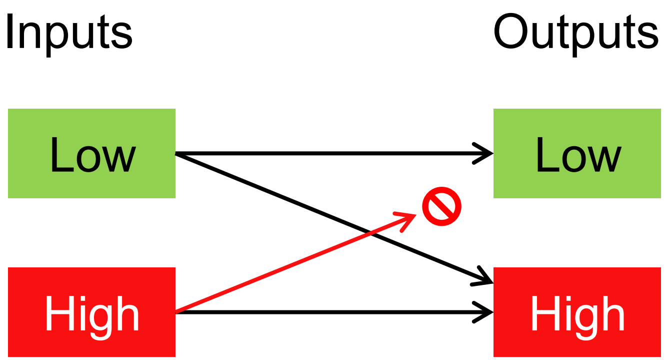
Security policies define the security requirements for a given system (Goguen and Meseguer, 1982)

Security policies on data usage:

- Confidentiality → data must not be accessed by unauthorized users
- Integrity → data must not be modified by unauthorized users
- Availability → data must come from available sources

Allowed and forbidden information flow

- Low input can flow to low and high output
- High input can flow to high output, but **NOT** to low output



Trace Properties (1)

A trace property is…

- … *the intersection of a safety property and a liveness property*. (Alpern and Schneider, 1985)
  - Safety property → “bad things” do not happen
  - Liveness property → “a good thing” happens

- … *a predicate on a single system execution*. (Kozyri et al., 2022)

Trace Properties (2)

Can be implemented through, e.g.:

- Access Control:
  - E.g. Bell–La Padula model (Bell and La Padula, 1973)
    - Low subject must not read from higher objects
    - High subject must not write to lower objects

- Encryption:
  - Hiding information from users without decryption key

Hyperproperties

- Trace properties cannot cover complex security policies
- System properties are derived from multiple traces
- Hyperproperties define properties on sets of traces (Clarkson and Schneider, 2010)
- Also cover trace properties

➔ Information Flow Properties

Information Flow Properties (1)

An **information flow property** is a mathematical specification of how information is allowed to flow between entities making up a system, such as programs, users, inputs, outputs, and storage locations. (Kozyri et al., 2022)

Can be specified for any abstraction level of a system:

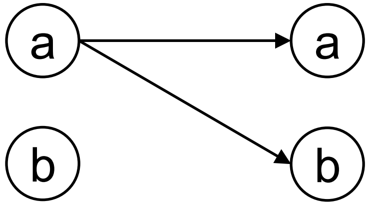
- Hardware
- Operating system
- Programming Language
- Distributed systems
- Cyber-physical systems

Information Flow Properties (2)

Explicit flow

- Direct transfer of information
- Assignments or deducing new values

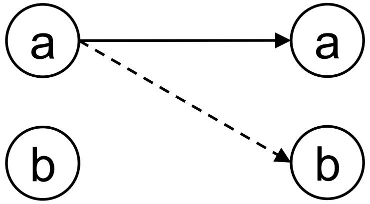
```
b := a
```



Implicit flow

- Indirect transfer of information
- E.g. through conditional statements

```
if a < 1 then {
    b := 0
} else {
    b := 1
}
```



Information Flow Properties (3)

Properties

- Strong dependency (Cohen, 1976)

  Given a system with input and output labels α and β.
  β depends strongly on α if β varies for two execution traces where only α differs.

- Noninterference (Goguen and Meseguer, 1982)
  - Prohibit information flow between certain entities
  - Relational noninterference (Kozyri et al., 2022):
    - Formalizes the relation between input and output labels
    - E.g., John’s co-workers cannot find out about his group memberships (secret output), but see his connections (public output)
  - Limitations :
    - Only for deterministic programs
    - Does not consider timing or termination

Information Flow Properties (4)

Other properties

- Nondeducibility (Sutherland, 1986)
  - No flow from high to low entities
  - Also no flow from low to high entities → symmetry (McLean, 1990)
    → too strict?
  - Relevant for properties, such as anonymity and unlinkability (Hughes and Shmatikov, 2004)

- Noninference (O'Halloran, 1990)
  - Removing high inputs and outputs results in valid traces
  - May be too strict → Generalized Noninference

Modelling Information Flow Properties (1)

Labels

- Assigning labels to entities, input data, output data, functions, etc.

- Granularity
    - Low-level: higher control, fine-grained, more complex
    - High-level: less control, more comprehensible

- Static binding:
    - Labelling of data containers
    - Can be done in advance, e.g., by the compiler

- Dynamic binding:
    - Labelling data values
    - Assignment of labels during runtime → Who assigns labels?

Modelling Information Flow Properties (2)

Flow relations

- Defining allowed data flow: $A \supseteq B$

- Axioms (Denning, 1976)

  Reflexive: $A \supseteq A$

  Transitive (Preorder): $A \supseteq B,\ B \supseteq C \ \Rightarrow\ A \supseteq C$

  Antisymmetric (Partial Order): $A \supseteq B,\ B \supseteq A \ \Rightarrow\ A = B$

Modelling Information Flow Properties (3)

- Joining labels
  - Example: `C:=A+B`

    ➔ What security class does C belong to?

- Linear ordering:
  - E.g., military system with hierarchical clearance levels

- Nonlinear ordering
  - E.g., system that contains medical, financial, and criminal records on individuals

Linear ordered lattice (from Denning, 1976)

$SC = \{A_1, \dots, A_n\}$

$A_i \rightarrow A_j \;\; iff \;\; i \leq j$

$A_i \oplus A_j \equiv A_{max}(i, j)$

$A_i \otimes A_j \equiv A_{min}(i, j)$

$L = A_1;\; H = A_n$

$$A_n \uparrow A_{n-1} \uparrow \dots \uparrow A_2 \uparrow A_1$$

Nonlinear lattice (from Denning, 1976)

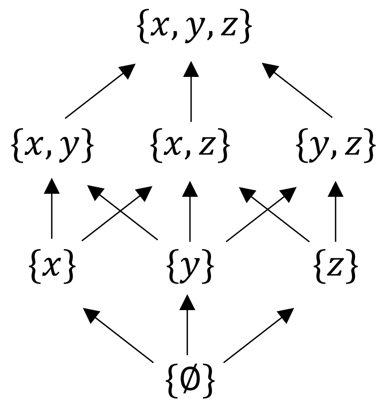
$SC = powerset(X)$

$A \rightarrow B \;iff\; A \subseteq B$

$A \oplus B \equiv A \cup B$

$A \otimes B \equiv A \cap B$

$L = \emptyset;\; H = X$



Determinism vs. Nondeterminism (1)

Deterministic system

- Same input always yields <u>the same output values</u>

Nondeterministic system

- Same input yields <u>varying output values</u>
- Set of output values can differ
- Distribution of output values can differ
- Occurs especially in connection with concurrency

Deterministic code:

```
if h < 1 then {
    l := 0
} else {
    l := 2
}
```

Nondeterministic code with different output set:

```
if h < 1 then {
    l := 0 || l := 1
} else {
    l := 0 || l := 2
}
```

Nondeterministic code with different output distribution:

```
if h < 1 then {
    l := 0 || l := 1 || l := 1
} else {
    l := 0 || l := 0 || l := 1
}
```

Determinism vs. Nondeterminism (2)

Nondeterministic information flow properties

- Observational determinism
  - Completely prohibits nondeterminism
- Possibilistic approach
  - Consider the set of possible outputs
  - Generalized noninterference (GNI) (McCullough, 1988)
- Probabilistic approach
  - Consider the probability
  - Probabilistic noninterference (Volpano and Smith, 1999)

Comparison of nondeterministic approaches (from Kozyri et al., 2022)

	Allows public nondeterminism	Defends against refinement (see slide 20)	Defends against leaky output distributions
Observational determinism	✗	✓	✓
Possibilistic	✓	✗	✗
Probabilistic	✓	?	✓

Attacks (1)

Covert channel attacks

- Leakage through illegitimate information channels
- Passive and active adversaries
- Examples: heat emission, program termination, time

Termination attack

- Confidential input may change termination behavior of a program
- Addressed by termination-sensitive noninterference (Volpano and Smith, 1997)

Varying termination behavior

```
if h < 1 then {
    while true do skip
} else {
    l := 1
}
```

Attacks (2)

Timing attacks

- Confidential input may change execution time of a program
- E.g., attacks against cryptographic algorithms
- Addressed by time-sensitive noninterference

Varying execution time

```
if h < 1 then {
    do(…)
    l := 1
} else {
    l := 1
}
```

Attacks (3)

Refinement attack

- Exploiting nondeterministic systems
- Manipulating system behavior to increase the likelihood for information leaks

- Example (from Clarkson and Schneider, 2010)

```
secret ∈ {0,1}

out:=0 || out:=1 || out:=secret
```

➔ Refinement: system that only outputs secret

Reclassification (1)

Objective

- Finer grained labels
- Changing the class of information from high to low or vice versa
- Practical examples:
  - Password checking
  - Publishing voting results
  - Encryption → confidentiality upgrade
  - Digital signature → integrity upgrade

Reclassification (2)

Types of reclassification (Confidentiality)

- Declassification
  - Escape hatch expressions → `declassify(x)`
  - E.g.: aggregated or anonymized data

- Erasure
  - Removing label for data to exclude class
  - E.g.: when the consent to information usage was withdrawn

Reclassification (3)

Types of reclassification (Integrity)

- Endorsement
  - Information becomes more trusted
  - E.g.: input sanitizing, or verifying entity

- Deprecation
  - Information becomes less trusted
  - E.g.: after a certain time period, or if processed by an untrusted entity

Reclassification (4)

Dimensions of declassification (Sabelfeld and Sands, 2009)

- What information can be released
  - Partial release / quantity → e.g. credit card number
- Who can release information
  - Related to integrity class of entities
- Where is information released
  - Level locality policies
  - Code locality policies
- When is information released
  - Time-complexity based: after a certain time
  - Probabilistic: likelihood of a leak
  - Relative: dependent on other events

Information Flow Control (1)

Information Flow Control ➔ Methods to enforce information flow properties

Static analysis

- Checking system behavior <u>before</u> execution
- Minimized runtime overhead
- Example approaches:
  - Static taint analysis
  - Security-typed languages, e.g. JFlow (Myers, 1999)

Information Flow Control (2)

Dynamic analysis

- Checking system behavior <u>during</u> execution
- Affects runtime performance
- May not detect implicit information flow (Myers and Liskov, 1997)
- Example approaches:
  - Dynamic taint analysis
  - Permissive-Upgrade (Austin and Flanagan, 2010)

Lecture Takeaways

- What are information flow properties?
- How to model them?
- What is noninterference?
- How can information flow be attacked?
- Why and how do we need reclassification?
- What are the differences between static and dynamic analysis?

Key Literature

- E. Kozyri, S. Chong, and A. C. Myers, ‘**Expressing Information Flow Properties**’, *FNT in Privacy and Security*, vol. 3, no. 1, pp. 1–102, 2022, doi: 10.1561/3300000008.
- D. E. Denning, ‘**A lattice model of secure information flow**’, *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, May 1976, doi: 10.1145/360051.360056.
- J. A. Goguen and J. Meseguer, ‘**Security Policies and Security Models**’, in *1982 IEEE Symposium on Security and Privacy*, Apr. 1982, pp. 11–11. doi: 10.1109/SP.1982.10014.
- R. Focardi and R. Gorrieri, ‘**Classification of Security Properties**’, in *Foundations of Security Analysis and Design*, R. Focardi and R. Gorrieri, Eds., Berlin, Heidelberg: Springer, 2001, pp. 331–396. doi: 10.1007/3-540-45608-2_6.

Further Literature

- Alpern and F. B. Schneider, '**Defining liveness**', *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, Oct. 1985, doi: 10.1016/0020-0190(85)90056-0.
- D. E. Bell and L. J. LaPadula, '**Secure computer systems: Mathematical foundations**', Citeseer, 1973. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a07edec9f865767be124e893c7a5a9547c8bf79e
- M. R. Clarkson and F. B. Schneider, '**Hyperproperties**', *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.
- E. S. Cohen, '**Strong Dependency: A Formalism for Describing Information Transmission in Computational Systems**', Dept. of Computer Science. Carnegie Mellon U., Pittsburg, Pa, 1976
- D. Sutherland, '**A model of information**', in *Proceedings of the 9th national computer security conference*, Washington, DC, 1986, pp. 175–183. Available: https://apps.dtic.mil/sti/tr/pdf/ADA221717.pdf#page=180
- C. O'Halloran, '**A calculus of information flow**', in Proceedings of the European Symposium on Research in Computer Security, 1990. Available: https://cir.nii.ac.jp/crid/1572824500057034240
- D. McCullough, '**Noninterference and the composability of security properties**', in Proceedings. 1988 IEEE Symposium on Security and Privacy, IEEE Computer Society, 1988, pp. 177–177. Available: https://www.computer.org/csdl/proceedings-article/sp/1988/08500177/12OmNzb7Ztw

Further Literature

- D. Volpano and G. Smith, '**Eliminating covert flows with minimum typings**', in *Proceedings 10th Computer Security Foundations Workshop*, Jun. 1997, pp. 156–168. doi: 10.1109/CSFW.1997.596807.
- D. Volpano and G. Smith, '**Probabilistic noninterference in a concurrent language**', Journal of Computer Security, vol. 7, no. 2–3, pp. 231–253, 1999. doi: 10.3233/JCS-1999-72-305.
- A. Sabelfeld and D. Sands, '**Declassification: Dimensions and principles**', *JCS*, vol. 17, no. 5, pp. 517–548, Oct. 2009, doi: 10.3233/JCS-2009-0352.
- A. C. Myers and B. Liskov, '**A decentralized model for information flow control**', *SIGOPS Oper. Syst. Rev.*, vol. 31, no. 5, pp. 129–142, Dec. 1997, doi: 10.1145/269005.266669.
- A. C. Myers, '**JFlow: practical mostly-static information flow control**', in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, San Antonio Texas USA: ACM, Jan. 1999, pp. 228–241. doi: 10.1145/292540.292561.
- T. H. Austin and C. Flanagan, '**Permissive dynamic information flow analysis**', in *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, Toronto Canada: ACM, Jun. 2010, pp. 1–12. doi: 10.1145/1814217.1814220.