

FOSAD Summer School 2015

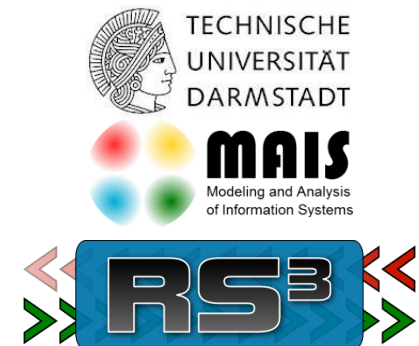
Concurrent Noninterference

Part 1: An Introduction to Noninterference

Heiko Mantel, Computer Science Department, TU Darmstadt

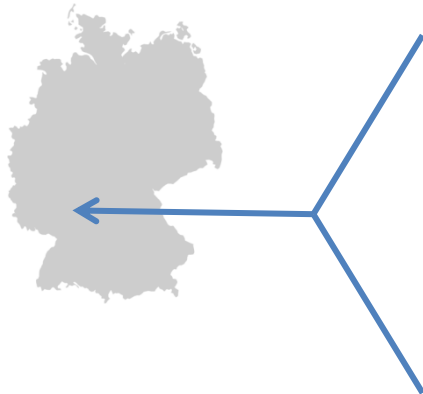
collaborators on this topic

Aslan Askarov, Timo Bähr, Steve Chong, Steffen Lortz,
Alexander Lux, Matthias Perner, Andrei Sabelfeld, David
Sands, Jens Sauer, David Schneider, Artem Starostin,
Henning Sudbrock, Alexandra Weber, ...



Where am I from?

at TU Darmstadt since 2007



the MAIS group



securing distributed systems

side channels

secure software engineering

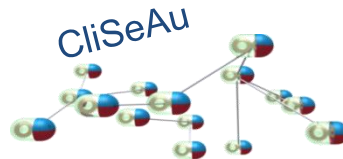
security analysis



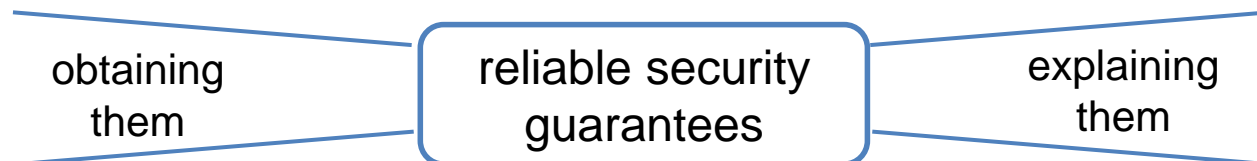
What are my research interests?

Current research interests

- reliable guarantees for software systems
 - focus: information-flow security and secure usage
 - analysis techniques and tools for deriving security guarantees
 - engineering techniques and tools for establishing security by design
 - languages for expressing security guarantees



- enable security-preserving abstraction, refinement, and composition
 - improving the understanding interplay between security and a, r, and c
 - side channel detection, analysis, and mitigation



My Current Research Projects



CROSSING



CASED



EC SPRIDE

EUROPEAN CENTER FOR
SECURITY AND PRIVACY BY DESIGN

we are
hiring

open post-doc positions

- ☐ concurrent program security
- ☐ information-flow security by design

open PhD positions

- ☐ concurrent program security
- ☐ mobile security

Roadmap

Part 1: An Introduction to Noninterference

- ❑ property-centric security vs mechanism-centric security
- ❑ information-flow security and information leakage
- ❑ noninterference: an informal definition
- ❑ a simple introduction to operational semantics
- ❑ noninterference: a formal definition
- ❑ example system: Cassandra
- ❑ exercises

Part 2: Noninterference for Multi-threaded Programs

Part 3: Recent Results on Concurrent Noninterference

Bibliography

Facets of Information Security

Security is CIA

C : Confidentiality

- ❑ *the nonoccurrence of unauthorized disclosure of information*

I : Integrity

- ❑ *the nonoccurrence of unauthorized modification of data or resources*

A : Availability

- ❑ *the degree to which a system or component is operational and accessible when required for use*

Other facets of security can be expressed using CIA

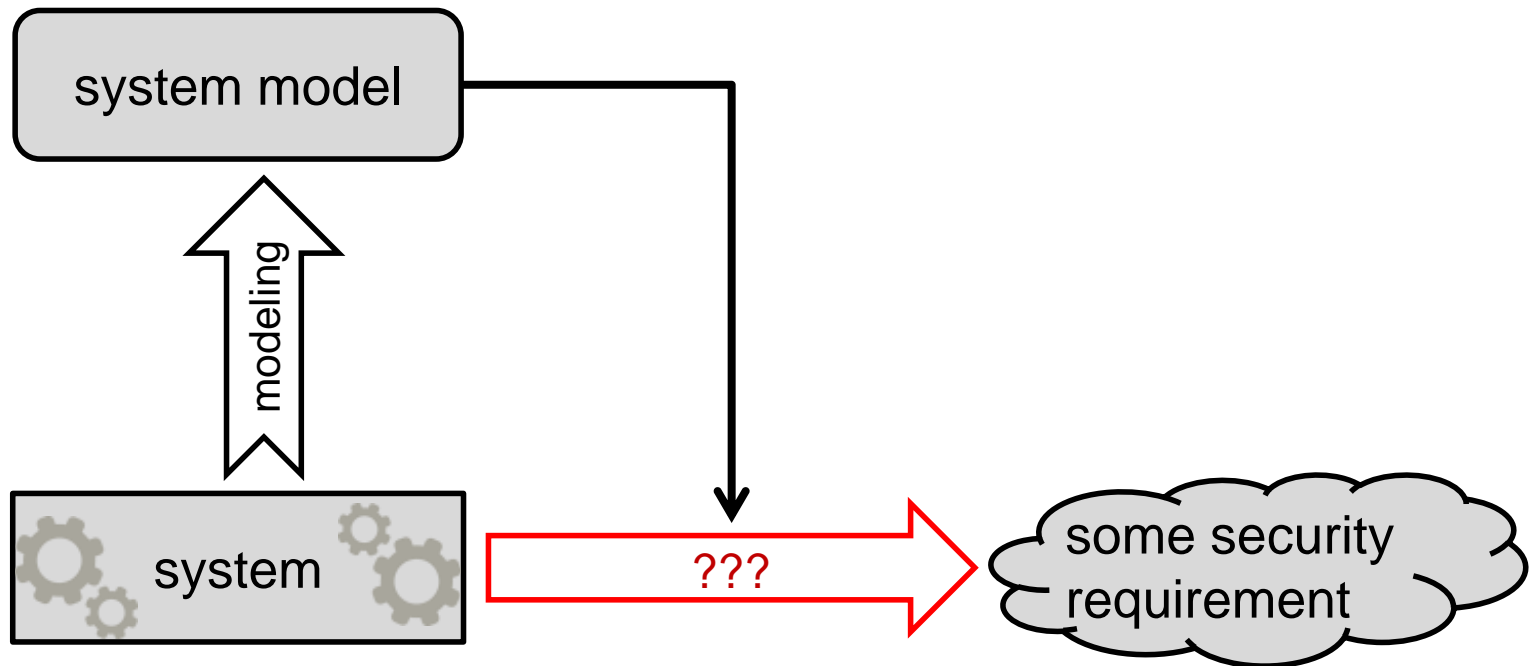
- ❑ e.g. anonymity, authenticity, non-reputability, privacy, ...

For instance, privacy can be expressed using CIA

- ❑ confidentiality of information you don't want to share means:
You choose what you let other people know.

Mechanism-centric Security (1)

Ensuring security



Is mechanism-centric security alone enough?

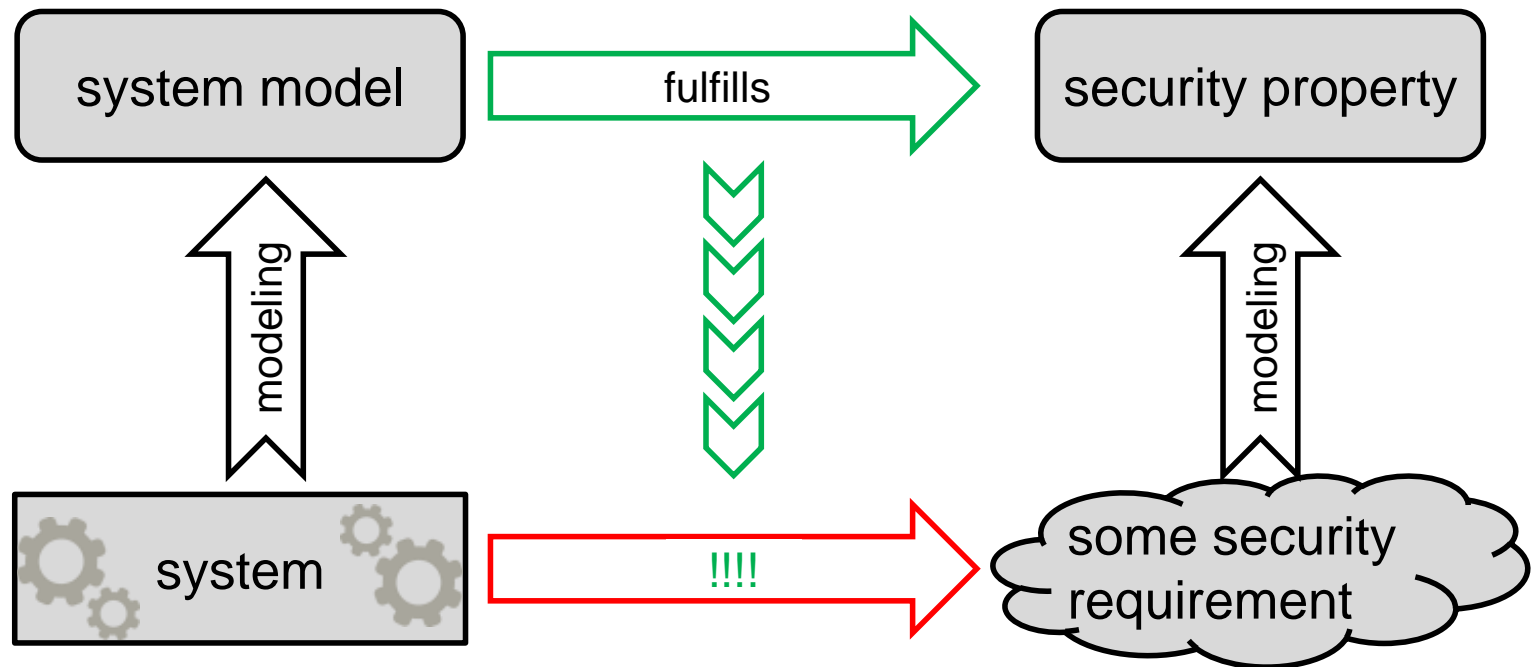
Mechanism-centric Security (2)



mechanism-
centric view
alone is not
enough

Property-centric Security

Modeling security as a property and then ensuring its satisfaction



Property-centric view should complement mechanism-centric one!

What is a property?

A property is an essential or distinctive attribute or quality of a thing.

Satisfaction of a property

- ❑ A system either has a given property or does not have it.
- ❑ If a system does not have a property, the system violates this property.

Example

A horse might satisfy the properties “... is fast.”, “... is brown.”, “... is big.”.

How to formulate a property?

- ❑ “... incorporates an access control mechanism” is a property, but the property-centric view does not provide added value for such a property
- ❑ “... is secure” nicely abstracts from security mechanism, but it is not a property, as a program might be “secure” for a user, but not for another

How to characterize conditions that are properties?

Characterizing Properties (1)

Properties can often be characterized by predicates on system runs.

Convention: Such a property is satisfied if the predicate holds for each run that this system could possibly perform.

How to characterize a property by a predicate on system runs?

- ❑ One defines a predicate P on individual runs, i.e. $P(\tau)$ holds or does not hold for a given system run τ .
- ❑ A system satisfies the property specified by P if and only if $P(\tau)$ holds for each run τ that is possible for this system.

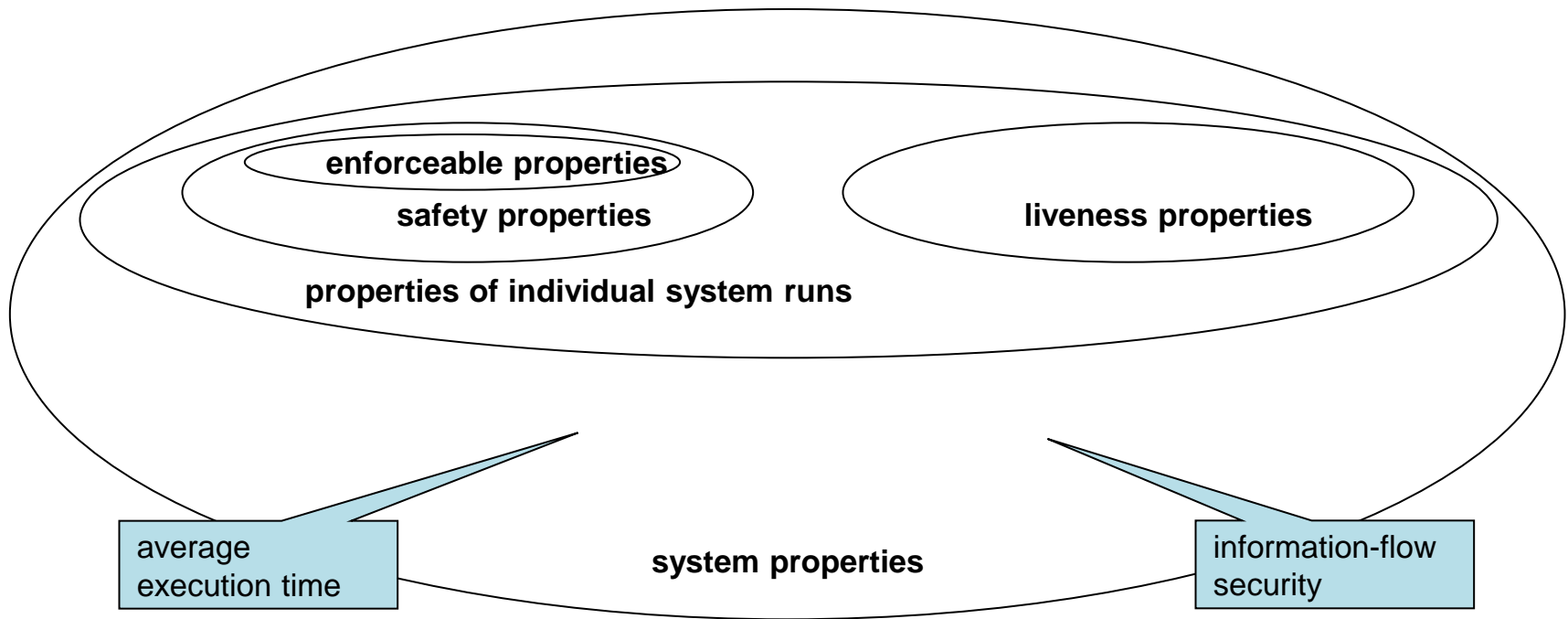
Example

- ❑ A system is terminating if each possible system run is finite

Many properties of interest in Computer Science can be characterized by predicates on system runs, but there are also properties that cannot be characterized in this way.

Characterizing Properties (2)

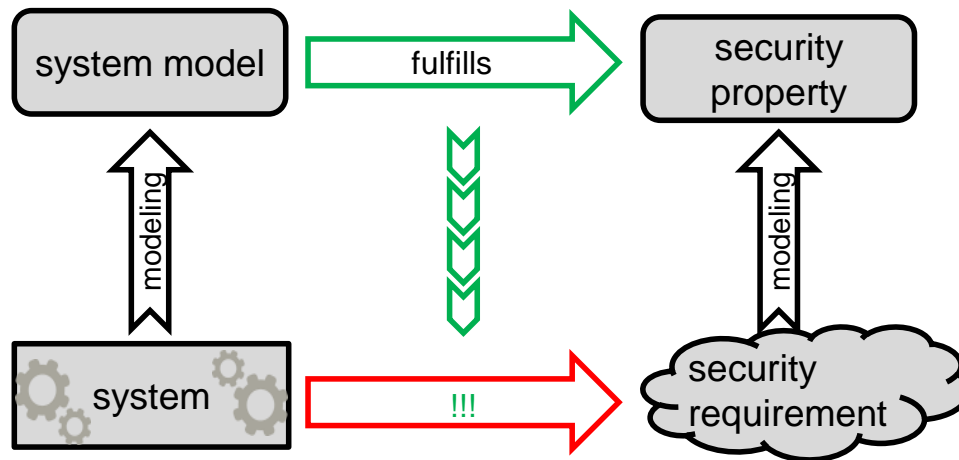
Properties can be classified according to their characterizations



Predicates on individual system runs are not expressive enough to characterize all security properties! (more by Michael Clarkson)

From Mechanisms to Properties

Property-centric security



Using security mechanisms to establish security properties

- ☐ access control \Rightarrow authorized accesses only
- ☐ usage control \Rightarrow secure usage
- ☐ information-flow control \Rightarrow information-flow security

The focus of this tutorial will be on information-flow security.

Why Information-Flow Security?

Example

- ☐ You install an app on your cell phone.
- ☐ How can you be sure that this app does not leak
 - ☐ your calendar,
 - ☐ your contacts,
 - ☐ your call history, or
 - ☐ your physical location?

Even if you feel OK with that an app leaks some of your private data, are you still OK if it leaks all of your private information in all cases?

- ☐ The purpose of information-flow security is to limit what data is leaked.

Note that access control is insufficient if the app's functionality needs access to your private data and also to information sinks where you don't want your private data to go.

Roadmap

Part 1: An Introduction to Noninterference

- ❑ property-centric security vs mechanism-centric security
- ❑ information-flow security and information leakage
- ❑ noninterference: an informal definition
- ❑ a simple introduction to operational semantics
- ❑ noninterference: a formal definition
- ❑ example system: Cassandra
- ❑ exercises

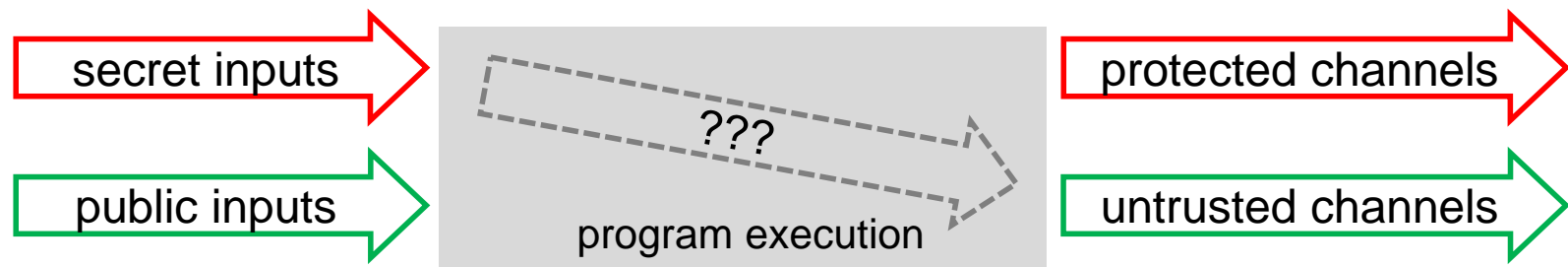
Part 2: Noninterference for Multi-threaded Programs

Part 3: Recent Results on Concurrent Noninterference

Bibliography

What is information-flow security? (1)

Is there any danger that secrets are leaked to untrusted sinks?



Information leakage

An attacker makes observations during a program run that allow him to deduce secret information.

Information-flow security

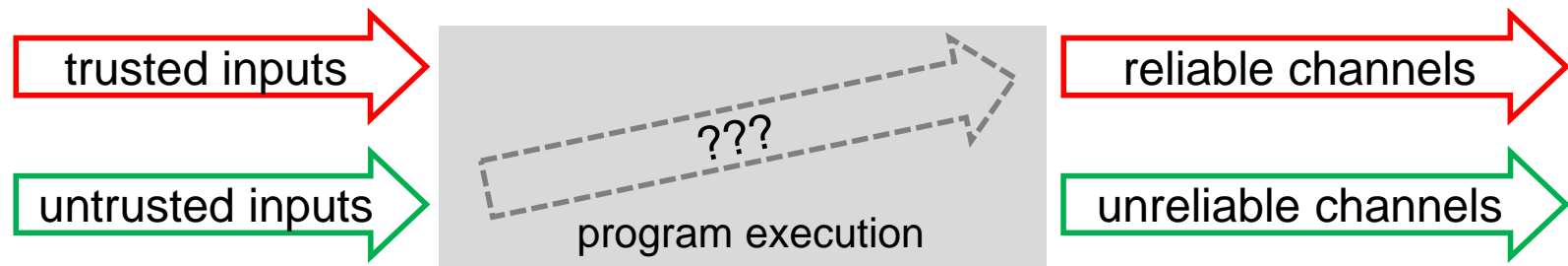
[for confidentiality]

There is no danger of information leakage.

Note: Information-flow security can also be understood as integrity.

What is information-flow security? (2)

Is there any danger of corruption?



Corruption

An attacker provides untrusted input that affects the output on reliable channels.

Information-flow security

[for integrity]

Output on reliable channels is no less trustworthy than trusted input.

In this tutorial, I will focus on confidentiality (previous slide).

Information Leakage

Might running this program leak secret information?

```
...  
x := exp  
...
```

Yes, if

- ☐ the value of the target x can be observed by the attacker
and
- ☐ the value of the expression exp depends on secrets
or whether the assignment is executed or not depends on secrets

Hence, for establishing information-flow security, one needs

- ☐ access control \Rightarrow Knowing what the attacker can observe.
- ☐ data-flow analysis \Rightarrow ... if values of expressions depend on secrets.
- ☐ control-flow analysis \Rightarrow ... if reachability of statements depends on ..

How might information leak? (1)

Explicit leakage (or: direct leakage)

output **secret** to **untrusted-channel**

attacker learns initial
value of **secret**

A secret is leaked to an untrusted channel assuming

- ❑ variable **secret** contains secret information when the statement is run
- ❑ attacker can observe messages sent to **untrusted-channel**

How does the attacker deduce secret information?

- ❑ When this program is run, the attacker observes some message **msg** .
- ❑ From this message and the program code, he deduces that the initial value of **secret** must have been the same as the value of **msg** .

How might information leak? (2)

Implicit leakage (or: indirect leakage)

attacker learns
whether initial value
of **secret** was positive

```
if secret > 0 then output 1 to untrusted-channel  
else output 0 to untrusted-channel
```

A secret is leaked to an untrusted channel assuming

- ❑ variable **secret** contains secret information when the statement is run
- ❑ attacker can observe messages sent to **untrusted-channel**

How does the attacker deduce secret information?

- ❑ When this program is run, the attacker observes some message **msg**.
- ❑ If value of **msg** is 1 then, he deduces that the initial value of **secret** must have been greater than zero. If value of **msg** is 0 then, the initial value of **secret** must have been smaller or equal than zero.

Conservative assumption: The attacker knows the program code.

How might information leak? (3)

Information leakage via non-termination

```
while secret  $\geq$  0 do skip od;  
output 1 to untrusted-channel
```

if attacker sees 1
then he learns that
the initial value of
secret was negative

Information leakage via non-progress

```
public := 0;  
while true do  
  output public to untrusted-channel;  
  if public < secret  
    then public := public+1  
    else while true do skip od  
  fi; od
```

attacker learns initial
value of **secret** from
the last output that
he sees in a run

How might information leak? (4)

Information leakage via array look-up

```
output public-array[secret] to untrusted-channel ;  
for i := 0 to length(public-array) do  
  output public-array[i] to untrusted-channel ;  
od
```

attacker can narrow down the initial value of **secret** to those positions where the value output equals the first output

Information leakage via array modification

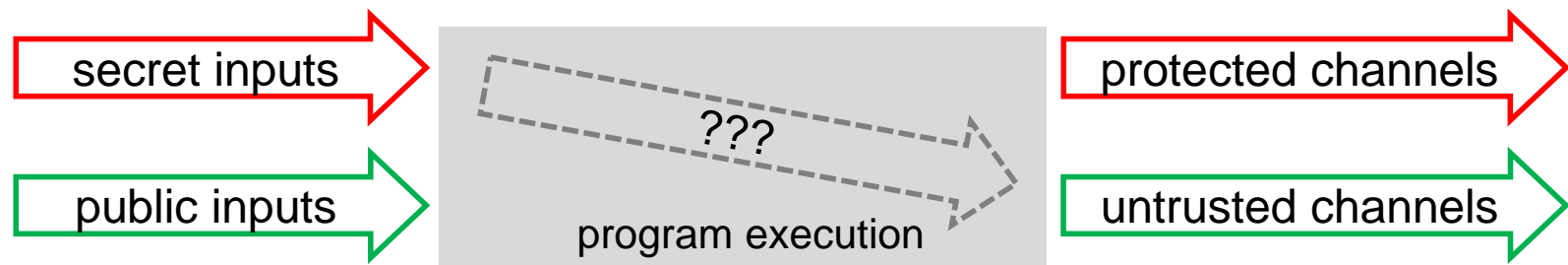
```
public-array[secret] := 42;  
for i := 0 to length(public-array) do  
  output public-array[i] to untrusted-channel ;  
od
```

attacker can narrow down the initial value of **secret** to those positions where the value 42 is output

Many further possibilities for leaking information exist, e.g., via dynamic dispatch, via exceptions, due to concurrency, ...

What is information-flow security? (3)

Is there any danger that secrets are leaked to untrusted sinks?



Information leakage

An attacker makes observations during a program run that allow him to deduce secret information.

Information-flow security

[for confidentiality]

There is no danger of information leakage.

Definition of information-flow security depends on what is secret, what the attacker can observe, and what the attacker can deduce.

Confidentiality and Attacker Models

Information that one might want to keep confidential

- ❑ initial values of dedicated program variables
- ❑ input on dedicated channels provided during a program run
- ❑ strategies used to determine the next input on dedicated channels
- ❑ ...

Capabilities of attackers that an attacker model could cover:

- ❑ attacker knows the program code
- and
- ❑ attacker observes final result of a program run
 - ❑ attacker observes output occurring during a program run
 - ❑ attacker observes intermediate values of variables during a run

More powerful attacker models are possible (e.g. timing, power).

Roadmap

Part 1: An Introduction to Noninterference

- ❑ property-centric security vs mechanism-centric security
- ❑ information-flow security and information leakage
- ❑ noninterference: an informal definition
- ❑ a simple introduction to operational semantics
- ❑ noninterference: a formal definition
- ❑ example system: Cassandra
- ❑ exercises

Part 2: Noninterference for Multi-threaded Programs

Part 3: Recent Results on Concurrent Noninterference

Bibliography

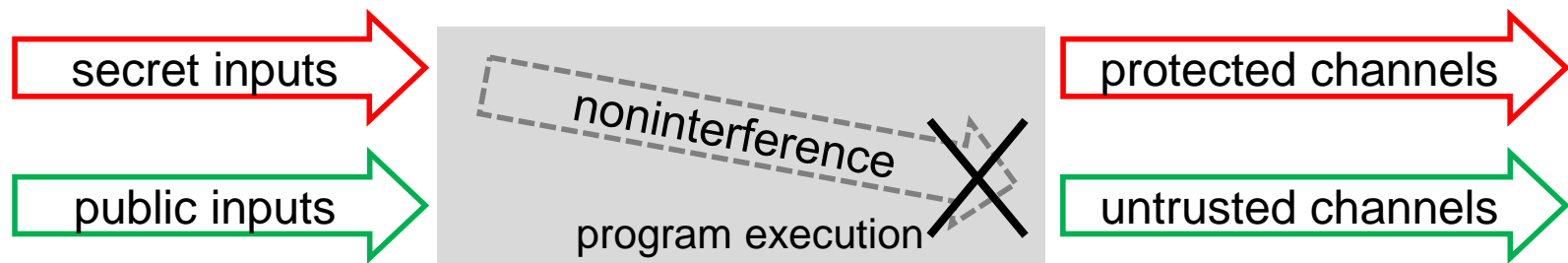
What is noninterference? (1)

Noninterference informally

A program is noninterferent if the observations that an attacker makes during runs of this program do not depend on secrets in any way.

Why does noninterference characterize information-flow security?

When a noninterferent program is run, the attacker makes observations. Since his observations do not depend on secrets, the attacker does not know more secrets after the run than before, i.e., no information is leaked.



Definition of noninterference depends on what is secret, what the attacker can observe, and how “dependence” is defined.

What is noninterference? (2)

Noninterference informally (like before)

A program is noninterferent if the observations that an attacker makes during runs of this program do not depend on secrets in any way.

Example

Is the following program noninterferent?

```
if secret > 0 then output 1 to untrusted-channel  
else output 0 to untrusted-channel
```

Better: Is the program noninterferent if the initial value of **secret** is a secret and the attacker can observe messages on **untrusted-channel** .

Answer: Under these conditions, the program is **not** noninterferent.

Argument: Which value the attacker observes on **untrusted-channel** during a program run depends on the initial value of **secret** .

That is, the attackers observations depend on a secret.

Here, semantics of programming language is relevant

Ensuring Noninterference

Possibilities for verifying that a program is noninterferent

- ❑ direct verification using the unwinding technique
- ❑ dedicated program analysis techniques (different traditions exist: type systems, program dependence graphs, abstract interpretation, ...)
- ❑ dedicated program logics
- ❑ general-purpose program logics (using self composition)

Possibilities for enforcing noninterference

- ❑ program transformations
- ❑ dynamic program analysis techniques (attention: some pitfalls)
- ❑ hybrid analysis techniques (combine static and dynamic analysis)

For verifying the soundness of such verification and enforcement techniques our definition of noninterference is too imprecise.

Roadmap

Part 1: An Introduction to Noninterference

- ❑ property-centric security vs mechanism-centric security
- ❑ information-flow security and information leakage
- ❑ noninterference: an informal definition
- ❑ a simple introduction to operational semantics
- ❑ noninterference: a formal definition
- ❑ example system: Cassandra
- ❑ exercises

Part 2: Noninterference for Multi-threaded Programs

Part 3: Recent Results on Concurrent Noninterference

Bibliography

Syntax

Syntax of a programming language

- A formally defined language.

Example

- $prog := stop \mid skip \mid x := aexp$
 $\mid \text{input } x \text{ from } ch \mid \text{output } aexp \text{ to } ch$
 $\mid prog; prog \mid \text{if } bexp \text{ then } prog \text{ else } prog \text{ fi}$
 $\mid \text{while } bexp \text{ do } prog \text{ od}$
- $aexp \in AExp$ “arithmetic expressions”
- $bexp \in BExp$ “boolean expressions”
- $x \in Var$ “program variables”
- $ch \in Ch$ “communication channels”

You will have a good intuition about the meaning of such programs.

Operational semantics gives a precise meaning to programs.

Operational Semantics (1)

Memory state

A memory state is a function $mem: Var \rightarrow Val$.

$mem(x)$ is the value of the variable $x \in Var$ in the memory state mem .

Evaluation of arithmetic expressions

$\langle aexp, mem \rangle \Downarrow n$ models that the arithmetic expression $aexp$ evaluates to the number $n \in Val$ in the memory state mem .

Evaluation of boolean expressions

$\langle bexp, mem \rangle \Downarrow b$ models that the boolean expression $bexp$ evaluates to the boolean value $b \in \{true, false\}$ in the memory state mem .

We leave the syntax and semantics of arithmetic expressions and of boolean expressions otherwise unspecified for now.

Operational Semantics (2)

Configuration

$$\langle prog, mem \rangle \in Prog \times Mem$$

- where $Prog$ is the set of all programs and
- where $Mem = Var \rightarrow Val$ is the set of all memory states

Intuition

A configuration $\langle prog, mem \rangle$ models a snapshot during a program run,

- where $prog$ models the program that remains to be executed and
- where mem models the current values of all program variables.

Example

$$\langle x := 42; y := x * y, [x \mapsto 0, y \mapsto 0] \rangle$$

- Two assignments remain to be executed.
- Both variables currently have value 0.

Operational Semantics (3)

Labeled transitions capturing computation steps

$$\langle prog, mem \rangle \xrightarrow{\alpha} \langle prog', mem' \rangle$$

- where $\langle prog, mem \rangle$ and $\langle prog', mem' \rangle$ are configurations and
- where $\alpha \in Ev$ is an event (Ev remains unspecified for now)

Intuition

A transition $\langle prog, mem \rangle \xrightarrow{\alpha} \langle prog', mem' \rangle$ captures a computation step

- where $\langle prog, mem \rangle$ is the configuration before the step,
- where $\langle prog', mem' \rangle$ is the configuration after the step, and
- where the event α captures additional information, e.g. a value output

Example

$$\langle x := 42; y := x * y, [x \mapsto 0, y \mapsto 0] \rangle \xrightarrow{\cdot} \langle y := x * y, [x \mapsto 42, y \mapsto 0] \rangle$$

Operational Semantics (4)

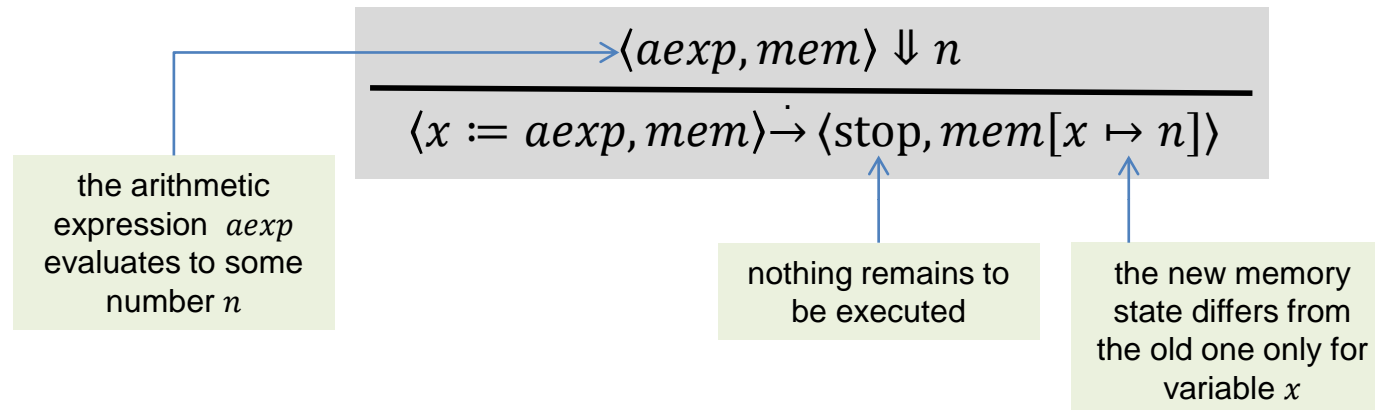
Form of derivation rules

$$\frac{\text{premise 1} \cdots \text{premise } n}{\text{conclusion}}$$

Intuition

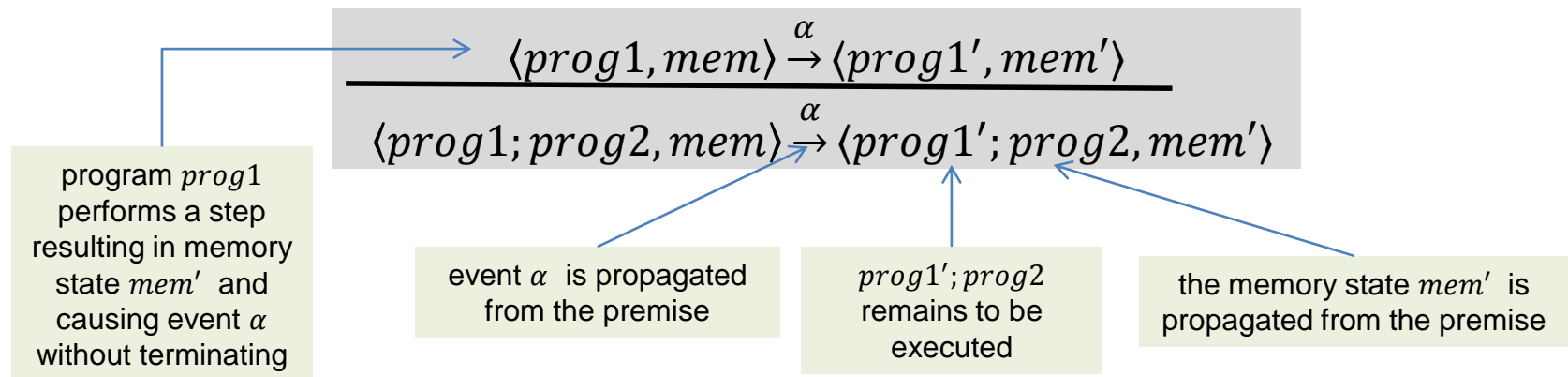
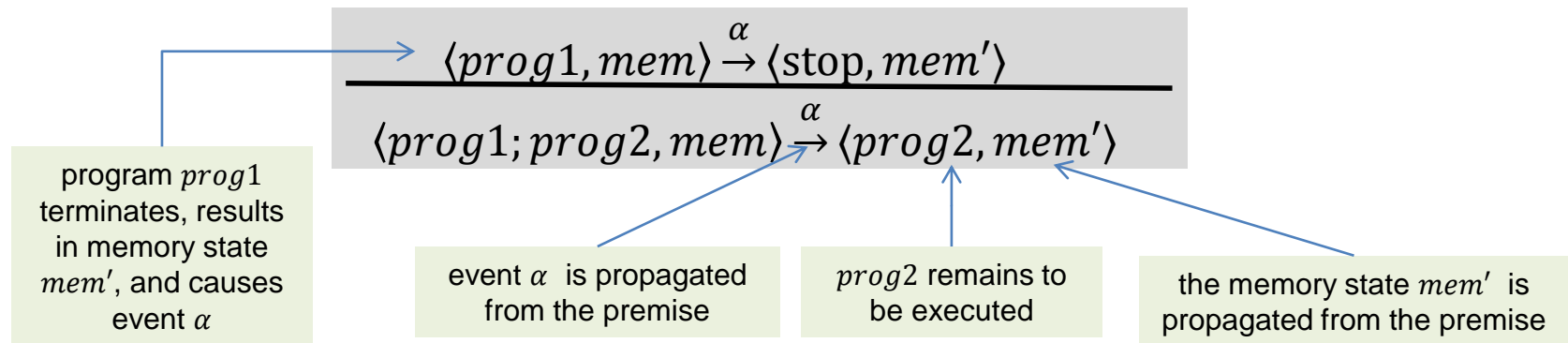
If each premise is true then the conclusion is also true.

A derivation rule for assignments



Operational Semantics (5)

Derivation rules for sequential composition



Operational Semantics (6)

A derivation

$$\begin{array}{c}
 \hline
 \langle 42, [x \mapsto 0, y \mapsto 0] \rangle \Downarrow 42 \\
 \hline
 \langle x := 42, [x \mapsto 0, y \mapsto 0] \rangle \dot{\rightarrow} \langle \text{stop}, [x \mapsto 42, y \mapsto 0] \rangle \\
 \hline
 \langle x := 42; y := x * y, [x \mapsto 0, y \mapsto 0] \rangle \dot{\rightarrow} \langle y := x * y, [x \mapsto 42, y \mapsto 0] \rangle
 \end{array}$$

The rules applied in this derivation

$$\frac{\langle \text{prog1}, \text{mem} \rangle \xrightarrow{\alpha} \langle \text{stop}, \text{mem}' \rangle}{\langle \text{prog1}; \text{prog2}, \text{mem} \rangle \xrightarrow{\alpha} \langle \text{prog2}, \text{mem}' \rangle}$$

$$\frac{\langle \text{aexp}, \text{mem} \rangle \Downarrow n}{\langle x := \text{aexp}, \text{mem} \rangle \dot{\rightarrow} \langle \text{stop}, \text{mem}[x \mapsto n] \rangle}$$

$$\frac{n \in \text{Val}}{\langle n, \text{mem} \rangle \Downarrow n}$$

a rule for evaluating arithmetic expressions that are constants

Operational Semantics (7)

Rules for conditionals and loops

$$\langle bexp, mem \rangle \Downarrow false$$

$$\frac{}{\langle \text{if } bexp \text{ then } prog1 \text{ else } prog2 \text{ fi}, mem \rangle \dot{\rightarrow} \langle prog2, mem \rangle}$$

$$\langle bexp, mem \rangle \Downarrow true$$

$$\frac{}{\langle \text{if } bexp \text{ then } prog1 \text{ else } prog2 \text{ fi}, mem \rangle \dot{\rightarrow} \langle prog1, mem \rangle}$$

$$\langle bexp, mem \rangle \Downarrow false$$

$$\frac{}{\langle \text{while } bexp \text{ do } prog \text{ od}, mem \rangle \dot{\rightarrow} \langle stop, mem \rangle}$$

$$\langle bexp, mem \rangle \Downarrow true$$

$$\frac{}{\langle \text{while } bexp \text{ do } prog \text{ od}, mem \rangle \dot{\rightarrow} \langle prog; \text{while } bexp \text{ do } prog \text{ od}, mem \rangle}$$

Operational Semantics (8)

Rule for skip

$$\frac{}{\langle \text{skip}, \text{mem} \rangle \dot{\rightarrow} \langle \text{stop}, \text{mem} \rangle}$$

Rules for output

$$\frac{\langle aexp, \text{mem} \rangle \Downarrow n}{\langle \text{output } aexp \text{ to } ch, \text{mem} \rangle \xrightarrow{\text{output}(ch, n)} \langle \text{stop}, \text{mem} \rangle}$$

$$\frac{}{\langle \text{input } x \text{ from } ch, \text{mem} \rangle \xrightarrow{\text{input}(ch, n)} \langle \text{stop}, \text{mem}[x \mapsto n] \rangle}$$

Operational Semantics (9)

Labeled transitions capturing runs

$$\langle prog, mem \rangle \xRightarrow{\tau} \langle prog', mem' \rangle$$

- where $\langle prog, mem \rangle$ and $\langle prog', mem' \rangle$ are configurations and
- where $\tau \in Ev^*$ is a sequence of events

Rules

$$\frac{}{\langle prog, mem \rangle \xRightarrow{\square} \langle prog, mem \rangle}$$

$$\frac{\langle prog, mem \rangle \xrightarrow{\alpha} \langle prog', mem' \rangle \quad \langle prog', mem' \rangle \xRightarrow{[\alpha_1, \dots, \alpha_n]} \langle prog'', mem'' \rangle}{\langle prog, mem \rangle \xRightarrow{[\alpha, \alpha_1, \dots, \alpha_n]} \langle prog'', mem'' \rangle}$$

Some Exercises (Homework 1)

1. Formalize the operational semantics of `repeat prog until bexp` by 2 rules.
2. Define sets of arithmetic and boolean expressions (i.e. $AExp$ and $BExp$) and rules for $\langle aexp, mem \rangle \Downarrow n$ and $\langle bexp, mem \rangle \Downarrow b$.

End of presentation on August 31

break & time for homework



Roadmap

Part 1: An Introduction to Noninterference

- ❑ property-centric security vs mechanism-centric security
- ❑ information-flow security and information leakage
- ❑ noninterference: an informal definition
- ❑ a simple introduction to operational semantics
- ❑ noninterference: a formal definition
- ❑ example system: Cassandra
- ❑ exercises

Part 2: Noninterference for Multi-threaded Programs

Part 3: Recent Results on Concurrent Noninterference

Bibliography

Formalizing Noninterference (1)

Noninterference informally (like before)

A program is noninterferent if the observations that an attacker makes during runs of this program do not depend on secrets in any way.

Choice: What is secret?

There is a dedicated set of variables $\text{high} \subseteq \text{Var}$. The initial values of these variables must be kept confidential.

Choice: What can the attacker observe?

There is a dedicated set of variables $\text{low} = \text{Var} \setminus \text{high}$. The initial and final values of these variables are what the attacker observes.

How to formally define noninterference for these choices?

Formalizing Noninterference (2)

Choice: What is secret? (like before)

There is a dedicated set of variables **high** $\subseteq \text{Var}$. The initial values of these variables must be kept confidential.

Choice: What can the attacker observe? (like before)

There is a dedicated set of variables **low** $= \text{Var} \setminus \text{high}$. The initial and final values of these variables are what the attacker observes.

Indistinguishability of memories for the attacker

Two memories $mem, mem': \text{Var} \rightarrow \text{Val}$ are indistinguishable (denoted by $mem =_{\text{low}} mem'$) if and only if

$$\forall x \in \text{low}. mem(x) = mem'(x)$$

Formalizing Noninterference (3)

Choice: What is secret? (like before)

There is a dedicated set of variables **high** $\subseteq \text{Var}$. The initial values of these variables must be kept confidential.

Choice: What can the attacker observe? (like before)

There is a dedicated set of variables **low** $= \text{Var} \setminus \text{high}$. The initial and final values of these variables are what the attacker observes.

A formal definition of noninterference for these choices

A program *prog* is noninterferent if and only if

$$\begin{aligned} & \forall mem1, mem2, mem1', mem': Var \rightarrow Val . \forall \tau, \tau' \in Ev^* . \\ & [\langle prog, mem1 \rangle \xRightarrow{\tau} \langle stop, mem2 \rangle \wedge mem1' =_{\text{low}} mem1] \\ & \Rightarrow [\langle prog, mem1' \rangle \xRightarrow{\tau'} \langle stop, mem2' \rangle \Rightarrow mem2' =_{\text{low}} mem2] \end{aligned}$$

Definition of noninterference is based on counter-factual reasoning.

Roadmap

Part 1: An Introduction to Noninterference

- ❑ property-centric security vs mechanism-centric security
- ❑ information-flow security and information leakage
- ❑ noninterference: an informal definition
- ❑ a simple introduction to operational semantics
- ❑ noninterference: a formal definition
- ❑ example system: Cassandra
- ❑ exercises

Part 2: Noninterference for Multi-threaded Programs

Part 3: Recent Results on Concurrent Noninterference

Bibliography

Nice Theory – Can it be applied?

Reference scenarios in the DFG priority program RS³:



www.spp-rs3.de

software security
for mobile devices

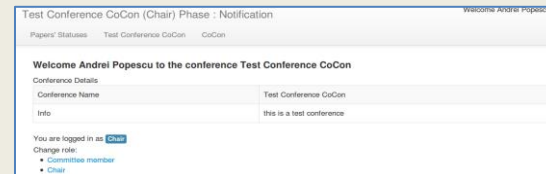
the RS³
Certifying
Appstore



see talk by David
Schneider today

security in web-based workflow
management systems

CoCon



security in E-voting

select
secure election



Some Exercises (Homework 2)

3. Argue why the program from Slide 27 does not satisfy our formal definition of noninterference.
4. Does the formal definition of noninterference on Slide 44 faithfully capture our informal definition of noninterference if
 - a. the attacker can observe the number of steps,
 - b. the attacker can observe interactions on the channel $open \in Ch$.
5. If your answer to 3a and/or 3b is NO, then modify the formal definition of noninterference such that it faithfully captures our informal definition of noninterference under the given conditions.
6. How to modify the operational semantics if the program environment chooses inputs on a channel based on prior interactions on this channel?

Some Lessons Learned

- ❑ How to complement the mechanism-centric view on security by a property-centric view in a useful way?
- ❑ What is information-flow security and what is it good for?
- ❑ How to capture information-flow security by noninterference?
- ❑ How to define noninterference formally?
 - ❑ based on the operational semantics of a programming language
- ❑ Which definition of noninterference is suitable depends on
 - ❑ which secrets need to be kept confidential,
 - ❑ what the attacker can observe, and
 - ❑ the model of execution.

Bibliography

Early definitions of noninterference-like properties (selected)

[Feiertag et al 1977] R.J. Feiertag, K.N. Levitt, L. Robinson: *Proving Multi-level Security of a System Design*. In: Proceedings of ACM Symposium on Operating Systems Principles. pp. 35-41 (1977)

[Cohen 1977] E. Cohen: *Information Transmission in Computational Systems*. In: Proceedings of ACM Symposium on Operating Systems Principles. pp. 133-139 (1977)

[Goguen/Meseguer 1982] J.A. Goguen, J. Meseguer: *Security Policies and Security Models*. In: Proceedings of IEEE Security and Privacy. pp. 11-20 (1982)

[Sutherland 1986] D. Sutherland: *A Model of Information*. In: Proceedings of National Computer Security Conference. (1986)

[McCullough 1987] D. McCullough: *Specifications for Multi-level Security and a Hook-Up Property*. In: Proceedings of IEEE Security and Privacy. pp. 161-166 (1987)

[Mantel 2011] H. Mantel: *Information Flow and Noninterference*. In: Encyclopedia of Cryptography and Security, 2nd ed. (2011)

Early information-flow analysis with or w/o soundness results (selected)

[Denning/Denning 1977] D.E. Denning, P.J. Denning: *Certification of Programs for Secure Information Flow*. In: Communications of the ACM 20(7). pp. 504-513 (1977)

[Goguen/Meseguer 1984] J.A. Goguen, J. Meseguer: *Unwinding and Inference Control*. In: Proceedings of IEEE Security and Privacy, pp. 75-87 (1984)

[Rushby 1992] J. Rushby: *Noninterference, Transitivity, and Channel-Control Security Policies*. TR CSL-92-02, SRI International. (1992)

[Volpano/Smith/Irvine 1996] D. Volpano, G. Smith, C. Irvine: *A Sound Type System for Secure Flow Analysis*. In: Journal of Computer Security 4(3). pp. 1-21 (1996)

[Myers/Liskov 1997] A.C. Myers, B. Liskov: *A Decentralized Model for Information Flow Control*. In: Proceedings of ACM Symposium on Operating Systems Principles. pp. 129-142 (1997)