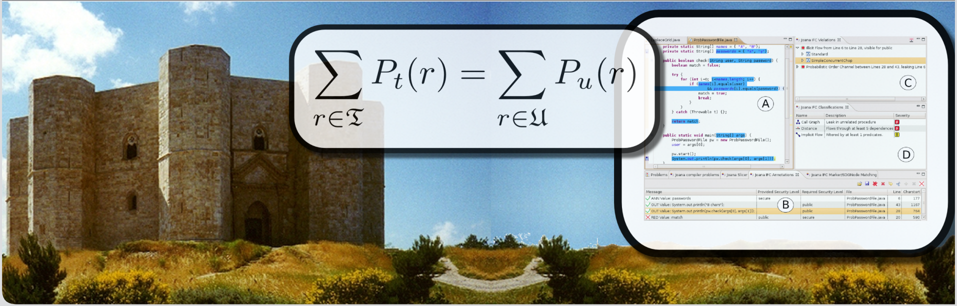
Karlsruher Institut für Technologie

X-Rays, not Passport Checks – Information Flow Control Using JOANA

Gregor Snelting

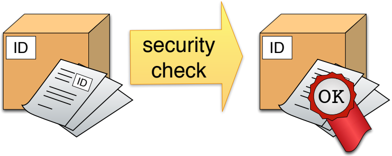
Presentation at SAP, 14.5.2014



Classical IT Security is not Enough!

Karlsruher Institut für Technologie

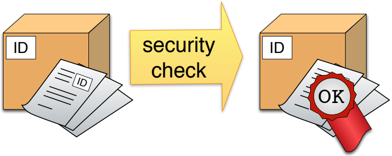
- classics: cryptography, certificates, intrusion detection, ...
  still necessary, but insufficient!
- classical approaches never analyse program code



- like passport checks – but passports can be faked

Classical IT Security is not Enough!

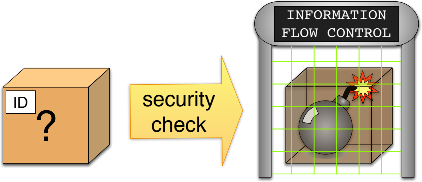
- classics: cryptography, certificates, intrusion detection, ...
  still necessary, but insufficient!
- classical approaches never analyse program code



- like passport checks – but passports can be faked
  Example 1: Stuxnet used stolen certificates
  Example 2: Heartbleed is based on an IFC problem

X-Rays, not Passport Checks!

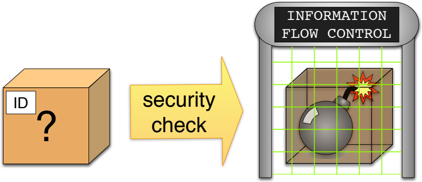
- Information Flow Control: analyse source / machine code, uncovers leaks and illegal information flow



X-Rays, not Passport Checks!

Karlsruher Institut für Technologie

- Information Flow Control: analyse source / machine code, uncovers leaks and illegal information flow

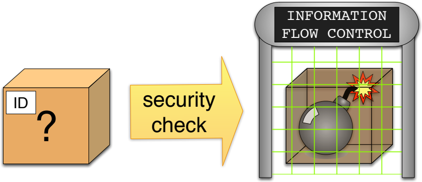


- advanced international research. Big projects: Mobius (EU), DFG SPP 1496 “Reliably Secure Software Systems”

X-Rays, not Passport Checks!

Karlsruher Institut für Technologie

- Information Flow Control: analyse source / machine code, uncovers leaks and illegal information flow



- advanced international research. Big projects: Mobius (EU), DFG SPP 1496 “Reliably Secure Software Systems”
- today: a few (!) useable tools

JOANA: Information Flow Control for Java
Download: `joana.ipd.kit.edu`

Information Flow Control (IFC)

Karlsruher Institut für Technologie

IFC analyses source/byte code, guarantees:

confidentiality: secret (“high”) values do not flow to public (“low”) ports
integrity: critical (“high”) computations not manipulated from outside (“low”)

Information Flow Control (IFC)

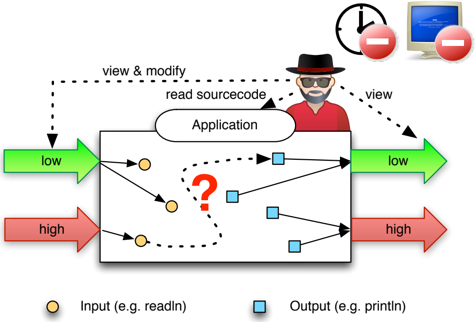
Karlsruher Institut für Technologie

IFC analyses source/byte code, guarantees:

confidentiality: secret (“high”) values do not flow to public (“low”) ports
integrity: critical (“high”) computations not manipulated from outside (“low”)

Assumptions:

- compiler, OS, hardware, ... are secure. IFC checks only application code!
- attacker knows code, can observe public output
- no physical side channels!



Confidentiality Leaks

Karlsruher Institut für Technologie

attacker gathers information about secret PIN:

```
void main():
  // inputPIN is high
  // print is low
  x = inputPIN();
  if (x < 1234)
     print(0);
  y = x;
  print(y);
```

explicit/implicit leaks

data or control flow depend on PIN

Confidentiality Leaks

attacker gathers information about secret PIN:

```
void main():
  // inputPIN is high
  // print is low
  x = inputPIN();
  if (x < 1234)
     print(0);
  y = x;
  print(y);
```

explicit/implicit leaks

data or control flow depend on PIN

```
void thread_1():
  // input is low
  x = input();
  print(x);

void thread_2():
  y = inputPIN();
  x = y;
```

possibilistic leak

some interleavings leak PIN

Confidentiality Leaks

attacker gathers information about secret PIN:

```
void main():
  // inputPIN is high
  // print is low
  x = inputPIN();
  if (x < 1234)
     print(0);
  y = x;
  print(y);
```

explicit/implicit leaks
data or control flow depend on PIN

```
void thread_1():
  // input is low
  x = input();
  print(x);

void thread_2():
  y = inputPIN();
  x = y;
```

possibilistic leak
some interleavings leak PIN

```
void thread_1():
  print("SA");

void thread_2():
  y = inputPIN();
  while (y != 0)
     y--;
  print("P");
```

probabilistic leak

Confidentiality Leaks

attacker gathers information about secret PIN:

```
void main():
  // inputPIN is high
  // print is low
  x = inputPIN();
  if (x < 1234)
     print(0);
  y = x;
  print(y);
```

explicit/implicit leaks

data or control flow depend on PIN

```
void thread_1():
  // input is low
  x = input();
  print(x);

void thread_2():
  y = inputPIN();
  x = y;
```

possibilistic leak

some interleavings leak PIN

```
void thread_1():
  print("SA");

void thread_2():
  y = inputPIN();
  while (y != 0)
     y--;
  print("P");
```

probabilistic leak

P(“*SAP*“) depends on PIN

IFC Technology

- theoretical security notion: (probabilistic) noninterference
- analysis methods: type systems, model checking, PDGs, ...

IFC Technology

- theoretical security notion: (probabilistic) noninterference
- analysis methods: type systems, model checking, PDGs, ...

Quality criteria:

- sound IFC guarantees to find all leaks!
  soundness proof [machine checked] required
- precise IFC generates few false alarms!
  sophisticated analysis algorithms required

IFC Technology

Karlsruher Institut für Technologie

- theoretical security notion: (probabilistic) noninterference
- analysis methods: type systems, model checking, PDGs, ...

Quality criteria:

- sound IFC guarantees to find all leaks!
  soundness proof [machine checked] required
- precise IFC generates few false alarms!
  sophisticated analysis algorithms required
  Remember Rice's Theorem: 100% sound and precise program analysis is undecideable

IFC Technology

Karlsruher Institut für Technologie

- theoretical security notion: (probabilistic) noninterference
- analysis methods: type systems, model checking, PDGs, ...

Quality criteria:

- sound IFC guarantees to find all leaks!
  soundness proof [machine checked] required
- precise IFC generates few false alarms!
  sophisticated analysis algorithms required
  Remember Rice's Theorem: 100% sound and precise program analysis is undecideable
- scaleable IFC analyses big programs!
  algorithm engineering required
- full-range IFC analyses full Java / C# / C++ !
  pointer analysis infrastructure required
- useable IFC needs little preprocessing!
  few annotations & nice GUI required

IFC Tools

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise

IFC Tools

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise 
- TAJ / Andromeda [Pistoia et al. 2009]: static analysis (part of IBM Security AppScan); full Java, high scalability, BUT moderately precise

IFC Tools

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise
- TAJ / Andromeda [Pistoia et al. 2009]: static analysis (part of IBM Security AppScan); full Java, high scalability, BUT moderately precise
- TaintDroid [Enck et al. 2010]: dynamic analysis; full Java, Android, application studies, BUT unsound, explicit flows (“taint”) only

IFC Tools

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise
- TAJ / Andromeda [Pistoia et al. 2009]: static analysis (part of IBM Security AppScan); full Java, high scalability, BUT moderately precise
- TaintDroid [Enck et al. 2010]: dynamic analysis; full Java, Android, application studies, BUT unsound, explicit flows (“taint”) only
- FlowDroid [Bodden 2013]: static analysis; no implicit flows, no probabilistic leaks, unsound, BUT Android apps & lifecycle

IFC Tools

Karlsruher Institut für Technologie

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise
- TAJ / Andromeda [Pistoia et al. 2009]: static analysis (part of IBM Security AppScan); full Java, high scalability, BUT moderately precise
- TaintDroid [Enck et al. 2010]: dynamic analysis; full Java, Android, application studies, BUT unsound, explicit flows ("taint") only
- FlowDroid [Bodden 2013]: static analysis; no implicit flows, no probabilistic leaks, unsound, BUT Android apps & lifecycle

IFC Tools

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise
- TAJ / Andromeda [Pistoia et al. 2009]: static analysis (part of IBM Security AppScan); full Java, high scalability, BUT moderately precise
- TaintDroid [Enck et al. 2010]: dynamic analysis; full Java, Android, application studies, BUT unsound, explicit flows ("taint") only
- FlowDroid [Bodden 2013]: static analysis; no implicit flows, no probabilistic leaks, unsound, BUT Android apps & lifecycle
- JOANA: static analysis; see below

IFC Tools

Karlsruher Institut für Technologie

- JIF [Myers et al 99]: static analysis; special language, many annotations, unprecise
- TAJ / Andromeda [Pistoia et al. 2009]: static analysis (part of IBM Security AppScan); full Java, high scalability, BUT moderately precise
- TaintDroid [Enck et al. 2010]: dynamic analysis; full Java, Android, application studies, BUT unsound, explicit flows (“taint”) only
- FlowDroid [Bodden 2013]: static analysis; no implicit flows, no probabilistic leaks, unsound, BUT Android apps & lifecycle
- JOANA: static analysis; see below

Do not confuse IFC tools with bug-finding tools (ESC/Java, Clousot, ...) !

- IFC tools find leaks, bug finders find null pointers, missing locks, ...
  many bug finders are scaleable (MLoc), but very unsound!

Noninterference

Karlsruher Institut für Technologie

- basic idea: public output is not influenced by secret data!
- sequential noninterference: for program Q, for all initial states s, s'

$$s \sim_{low} s' \implies [\![Q]\!]s \sim_{low} [\![Q]\!]s'$$

Noninterference

Karlsruher Institut für Technologie

- basic idea: public output is not influenced by secret data!
- sequential noninterference: for program Q, for all initial states s, s'

$$s \sim_{low} s' \implies [\![Q]\!]s \sim_{low} [\![Q]\!]s'$$

- for concurrent programs: treatment of nondeterminism?!
  idea: *probability* of public outputs is not influenced by secret data

Noninterference

Karlsruher Institut für Technologie

- basic idea: public output is not influenced by secret data!
- sequential noninterference: for program Q, for all initial states s, s'

$$s \sim_{low} s' \implies [\![Q]\!]s \sim_{low} [\![Q]\!]s'$$

- for concurrent programs: treatment of nondeterminism?!
  idea: *probability* of public outputs is not influenced by secret data
- Q is probabilistic noninterferent if

$$\sum_{t \in \mathfrak{T}} P_i(t) = \sum_{t \in \mathfrak{U}} P_{i'}(t)$$

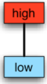
where $P_i(t)$ is the probability of trace t under input i, $\mathfrak{T}$ are the low-equivalent traces caused by i

JOANA in a Nutshell

```
void main() {
  int h = input();
  int l = encode(h);
  output(l);
}

int encode(int x) {
  if (x > 42)
    return 1;
  else
    return 0;
}
```

\+



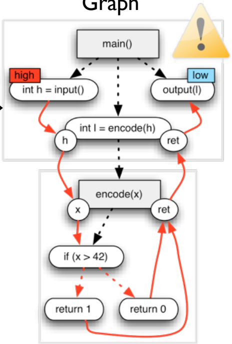
security lattice

\+

annotations

high `input()` low `output(_)`

System Dependence Graph



- full Java (up to 100kLOC)
- static whole program analysis
- applies program slicing
- applies points-to analysis
- flow-, context-, object-sensitive
- threads: probabilistic & possibilistic

Analysis Result

non-interference guarantee

or

possible leaks

Machine-checked proofs

- Classical non-interference with slicing
- Slicing theorem

∄ path a → b
⇒ definitely no information flow

∃ path a → b
⇒ information flow possible

JOANA Features

- sound
- full Java bytecode
- unlimited threads
- few false alarms
- few annotations
- declassifications
- Android Apps
- Eclipse plugin, webstart GUI
- open source

JOANA Features

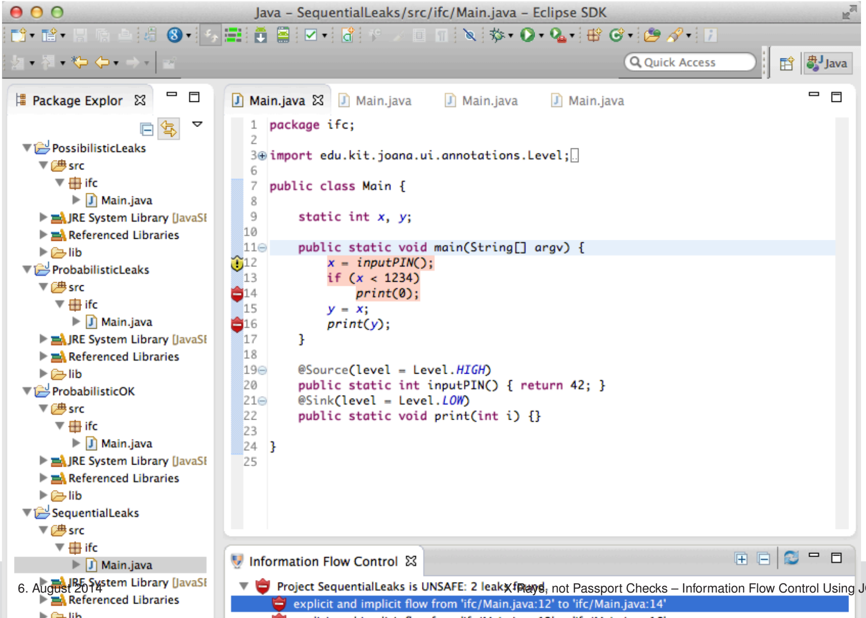
- sound
- full Java bytecode
- unlimited threads
- few false alarms
- few annotations
- declassifications
- Android Apps
- Eclipse plugin, webstart GUI
- open source
- max 100kLoc
- case studies

  e.g. HSQLDB (50kLOC Java): analysis time ≈ 1 day on PC
- scenario: analyse security kernels / critical components, not full OS!

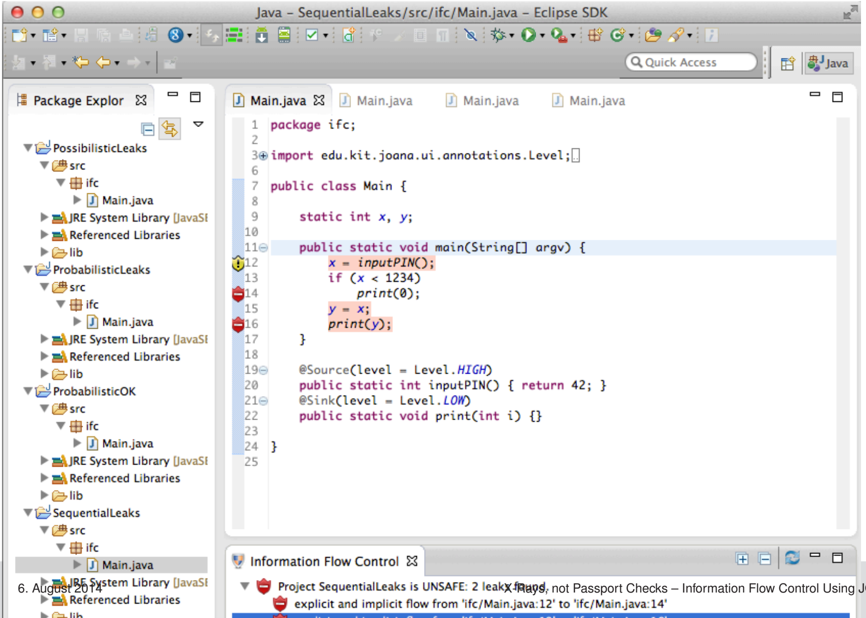
JOANA Demo

Jürgen Graf: Analysis of sequential & probabilistic leaks

Implicit Leak

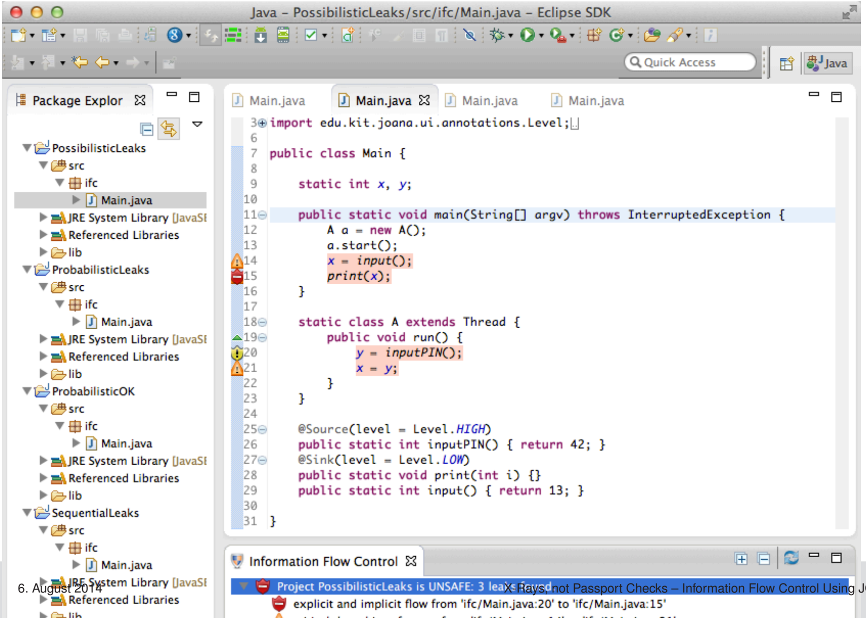


Explicit Leak

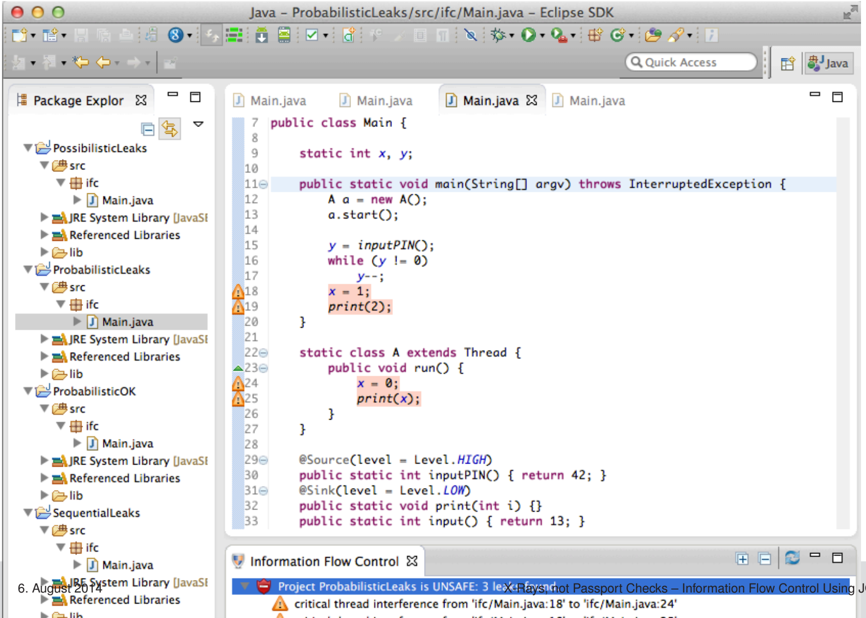


Possibilistic Leak

Karlsruher Institut für Technologie

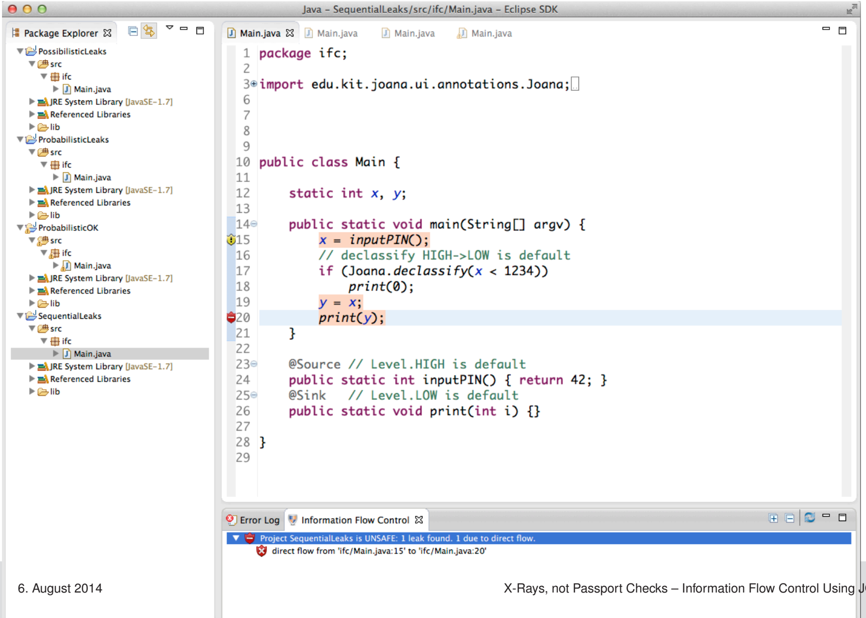


Probabilistic Leak



Declassification

Karlsruher Institut für Technologie



JOANA Technology

Karlsruher Institut für Technologie

- based on sophisticated program analysis:
  program dependence graphs (PDGs); exception-, pointer-, ... -analysis
- flow-, context-, object-, field-sensitive; optionally time-, lock-sensitive
  $\Rightarrow$ high precision, few false alarms

JOANA Technology

Karlsruher Institut für Technologie

- based on sophisticated program analysis:
  program dependence graphs (PDGs); exception-, pointer-, ... -analysis
- flow-, context-, object-, field-sensitive; optionally time-, lock-sensitive
  $\Rightarrow$ high precision, few false alarms
- (sequential) declassification in case noninterference is too strict
- machine-checked soundness proofs for sequential IFC

JOANA Technology

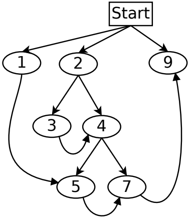
Karlsruher Institut für Technologie

- based on sophisticated program analysis:
  program dependence graphs (PDGs); exception-, pointer-, ... -analysis
- flow-, context-, object-, field-sensitive; optionally time-, lock-sensitive
  $\Rightarrow$ high precision, few false alarms
- (sequential) declassification in case noninterference is too strict
- machine-checked soundness proofs for sequential IFC
- for concurrent programs: new **RLSOD** algorithm
  [Relaxed Low-Security Observable Determinism]
  $\Rightarrow$ probabilistic noninterference without previous restrictions

A small PDG

Karlsruher Institut für Technologie

```
a = u();
while (f()) {
 x = v();
 if (x>0)
   b = a;
 else
   c = b;
}
z = c;
```



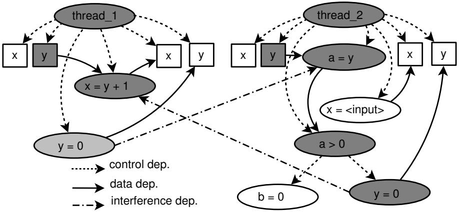
- $x \rightarrow y$: x controls execution of y; $x \rightsquigarrow y$: assigned var in x is used in y
- backward slice $BS(x) = \{y \mid y \rightarrow^* x\}$
- **Slicing Theorem.** [Reps et al 1988]
  Only statements/ expressions $y \in BS(x)$ can influence behaviour at x
- `u()` can influence `z`, `a` cannot influence `x>0`
- PDGs for full Java are nontrivial
  25 years of international research!

A multi-threaded PDG


```
int x, y;

void thread_1():
  x = y + 1;
  y = 0;
```

```
void thread_2():
  a = y;
  x = <input>;
  if a > 0
    b = 0;
  else
    y = 0;
```



- $BS(x) = \{y \mid y \to^*_{realizeable} x\}$
  "realizable": context- time- object-sensitive
  black: $BS("\mathtt{x} = \mathtt{y} + 1;")$; grey: time insensitive
- **Theorem.**[Snelting et al 2006] A program is (sequentially) noninterferent, if no high source is in backward slice of a low sink
  machine-checked proof: [Wasserrab 2009]

Conclusion

Karlsruher Institut für Technologie

- IFC today is practical: X-rays, not passport checks
- JOANA offers precise IFC for realistic Java programs
- JOANA contains groundbreaking algorithms + validation + proofs
- JOANA is open source
- JOANA was used in realistic case studies

Conclusion

Karlsruher Institut für Technologie

- IFC today is practical: X-rays, not passport checks
- JOANA offers precise IFC for realistic Java programs
- JOANA contains groundbreaking algorithms + validation + proofs
- JOANA is open source
- JOANA was used in realistic case studies
- new: JOANA handles pluggable (Android) components
- new: JOANA handles message encryption without declassification

Conclusion

Karlsruher Institut für Technologie

- IFC today is practical: X-rays, not passport checks
- JOANA offers precise IFC for realistic Java programs
- JOANA contains groundbreaking algorithms + validation + proofs
- JOANA is open source
- JOANA was used in realistic case studies
- new: JOANA handles pluggable (Android) components
- new: JOANA handles message encryption without declassification

JOANA is an achievement in IT security

JOANA main contributors:
G. Snelting, D. Giffhorn, J. Graf, C. Hammer, M. Hecker, J. Krinke, M. Mohr, D. Wasserrab

JOANA sponsors: DFG Sn11/5-1/2, DFG Sn11/9-1/2, DFG Sn11/11-1/2, DFG Sn11/12-1/2 [SPP 1496 “Reliably Secure Software Systems”], BMBF Center for Cyber Security KASTEL

JOANA papers: TOSEM 2006, IJIS 2009, PLAS 2009, CSF 2012, IT 2014, IJIS 2014, ...

LSOD

Karlsruher Institut für Technologie

Low-Security Observational Determinism [Roscoe] [Zdancewicz]:
low-equivalent inputs must generate low-equivalent traces

- $i \sim_{low} i'$, $\mathfrak{T}$ possible traces for i, $\mathfrak{U}$ possible traces for i'
  $\implies \forall T, U \in \mathfrak{T} \cup \mathfrak{U} : T \sim_{low} U$

  "the order of low events is not influenced by high events"

$\Rightarrow$ LSOD is scheduler independent
Theorem. [Zdancewic 2003]
LSOD guarantees probabilistic noninterference

LSOD

Low-Security Observational Determinism [Roscoe] [Zdancewicz]:
low-equivalent inputs must generate low-equivalent traces

- $i \sim_{low} i'$, $\mathfrak{T}$ possible traces for i, $\mathfrak{U}$ possible traces for i'
  $\implies \forall T, U \in \mathfrak{T} \cup \mathfrak{U} : T \sim_{low} U$

  "the order of low events is not influenced by high events"

⇒ LSOD is scheduler independent
Theorem. [Zdancewic 2003]
LSOD guarantees probabilistic noninterference

- BUT soundness problems / severe restrictions in early LSOD definitions
  ⇒ so far,other approaches more popular: Weak probabilistic noninterference [Volpano&Smith], Strong security [Sabelfeld&Sands], ...

NEW: RLSOD

Relaxed LSOD [Giffhorn 2012PhD, Giffhorn & Snelting 2013]:

- guarantees probabilistic noninterference
- avoids prohibition of secure low-nondeterminism
- precise: flow- context- object- field- time-sensitive
- soundness proof
- full Java, arbitrary threads (no reflection)
- scales up to 100kLOC
- succesful case studies [Küsters & Graf 2012, ...]

NEW: RLSOD

Relaxed LSOD [Giffhorn 2012PhD, Giffhorn & Snelting 2013]:

- guarantees probabilistic noninterference
- avoids prohibition of secure low-nondeterminism
- precise: flow- context- object- field- time-sensitive
- soundness proof
- full Java, arbitrary threads (no reflection)
- scales up to 100kLOC
- succesful case studies [Küsters & Graf 2012, ...]

Flow-sensitivity is the key! other ingredients:

- new definition for $T \sim_{low} U$ in case of nontermination
  $\Rightarrow$ no soundness leaks for infinite traces
  cave: RLSOD is termination-insensitive
- uses program dependence graphs (PDGs)
  $\Rightarrow$ sound & precise static approximation of RLSOD criterion

NEW: IFC and Crypto

Karlsruher Institut für Technologie

- so far, IFC cannot handle crypto (e.g. encrypted message passing)
  IFC needs declassification for crypto channels !?

$\Longrightarrow$ Küster's idea [CSF 2012]:

1. replace crypto code by stub which generates random numbers: $P \leadsto P'$
2. use JOANA to prove that P' is secure
3. Theorem: if P' secure, and P uses "perfect" crypto, then P secure ("noninterference guarantees computational indistinguishability w.r.t. unbounded adversaries")

$\Longrightarrow$ allows to apply JOANA to distributed systems, where components communicate via encrypted messages: e-voting, cloud storage

- recent work: Integration with KeY, extend for digital signatures and symmetric crypto ("CVJ" Projekt)