Secure Information Flow and Pointer Confinement in a Java-like Language

Anindya Banerjee and David A. Naumann

ab@cis.ksu.edu, naumann@cs.stevens-tech.edu

Kansas State University and Stevens Institute of Technology

www.cis.ksu.edu/~ab,

www.cs.stevens-tech.edu/~naumann

The Problem

- System with High and Low inputs, L $\leq$ H.
  H $\equiv$ secret/private/classified
- L users permitted to see L outputs.

(Security Policy: Confidentiality $\equiv$ “PROTECT SECRETS”)

Formalise for systems programmed in Java-like languages.

The Problem

- System with High and Low inputs, L $\leq$ H.
  H $\equiv$ secret/private/classified

- L users permitted to see L outputs.

(Security Policy: Confidentiality $\equiv$ "PROTECT SECRETS")

Formalise for systems programmed in Java-like languages.

- Noninterference (NI) [Goguen-Meseguer '82]

  "No matter how H inputs change, L outputs remain same".

$\equiv$ No information flow from H to L.

Our Contribution

Type-based analysis for secure information flow.

- Sequential, Java-like language
  - private fields, class-based visibility
  - mutually recursive classes, methods
  - pointers, mutable state, dynamic allocation
  - inheritance, dynamic dispatch

Our Contribution

Type-based analysis for secure information flow.

- Sequential, Java-like language
- Security type system

Our Contribution

Type-based analysis for secure information flow.

- Sequential, Java-like language
- Security type system
  - Data flow (via mutable fields)

Our Contribution

Type-based analysis for secure information flow.

- Sequential, Java-like language
- Security type system
  - Data flow (via mutable fields)
  - Control flow (via dynamic dispatch)

Our Contribution

Type-based analysis for secure information flow.

- Sequential, Java-like language
- Security type system
  - Data flow (via mutable fields)
  - Control flow (via dynamic dispatch)
- Proof of Noninterference (denotational semantics, compositional proofs)

What We Have Not Done

- Extension to full JavaCard
  - Exceptions
  - Protected fields, private/protected classes, interfaces, packages
- Extension to full Java
  - Threads
  - Class loading, Reflection, Native methods
  - Generics
  - ...

Previous Work: Main Inspirations

- Noninterference: Goguen-Meseguer, Denning-Denning
- Type-based analyses for information flow:

1996– Smith, Volpano (Simple Imperative Language)

1999– Abadi et al. (DCC – Info. flow as dependence analysis)

1999– Sabelfeld, Sands (Threads, Poss. NI, Prob. NI)

1999 Myers (Java – but NI open)

2000– Pottier, Simonet, Conchon (Core ML)

Previous Work: Main Inspirations

- Abstract Interpretation based analyses for info. flow:

1992 Mizuno, Schmidt (Logical relations to prove NI)

Previous Work: Main Inspirations

- Abstract Interpretation based analyses for info. flow:

  1992 Mizuno, Schmidt (Logical relations to prove NI)

Our focus:

- Type-based analyses for information flow: (Smith, Volpano)
  - Cope with threads, non-determinism, stochastic processes, declassification, ...
  - Cope with realistic programming languages.

Example: Aliasing (1)

```
class LPatient extends Object {//basic patient record
  String name;
  String getName() {return self.name;}
  unit setName(String n) {self.name := n;} }
```

Example: Aliasing (1)

```
class LPatient extends Object {//basic patient record
  String name;
  String getName() {return self.name;}
  unit setName(String n) {self.name := n;} }

class XPatient extends LPatient {
  String hiv;  //SECRET
  String getHIV() {return self.hiv;}
  unit setHIV(String s) {self.hiv := s;} }
```

Example: Aliasing (1)

```
LPatient lp := readFile();
String LBuf := lp.getName(); String HBuf := lp.getName();
```

LBuf ~ lp.name ~ HBuf

```
XPatient xp:= new XPatient();  xp.setName(LBuf);
```

LBuf ~ lp.name ~ HBuf ~ xp.name

Example: Aliasing (1)

```
LPatient lp := readFile();
String LBuf := lp.getName(); String HBuf := lp.getName();
```

LBuf ~ lp.name ~ HBuf

```
XPatient xp:= new XPatient();  xp.setName(LBuf);
```

LBuf ~ lp.name ~ HBuf ~ xp.name

```
String HBuf:= readFromTrustedChannel();  xp.setHIV(HBuf);
```

HBuf ~ xp.hiv;   LBuf ~ lp.name ~ xp.name

Example: Aliasing (1)

```
LPatient lp := readFile();
String LBuf := lp.getName(); String HBuf := lp.getName();
```

LBuf ~ lp.name ~ HBuf

```
XPatient xp:= new XPatient();  xp.setName(LBuf);
```

LBuf ~ lp.name ~ HBuf ~ xp.name

```
String HBuf:= readFromTrustedChannel();  xp.setHIV(HBuf);
```

HBuf ~ xp.hiv;   LBuf ~ lp.name ~ xp.name

```
LBuf := HBuf; lp.setName(xp.getHIV())
```

Example: Aliasing (1)

```
LPatient lp := readFile();
String LBuf := lp.getName(); String HBuf := lp.getName();
```

LBuf ~ lp.name ~ HBuf

```
XPatient xp:= new XPatient();  xp.setName(LBuf);
```

LBuf ~ lp.name ~ HBuf ~ xp.name

```
String HBuf:= readFromTrustedChannel();  xp.setHIV(HBuf);
```

HBuf ~ xp.hiv;   LBuf ~ lp.name ~ xp.name

```
LBuf := HBuf; lp.setName(xp.getHIV())
```

lp.name ~ xp.hiv

Annotated Types Prevent Direct Data Flows

```
class LPatient extends Object {
  (String, L) name;
  (String, L) getName() {return self.name;}
  (unit, L) setName((String, L) n) {self.name := n;} }
class XPatient extends LPatient {
  (String, H) hiv;  //SECRET
  (String, H) getHIV() {return self.hiv;}
  (unit, L) setHIV((String, H) s) {self.hiv := s; } }
```

No direct assignment from H to L

Example: Aliasing (1) Revisited

```
(LPatient, L) lp := readFile();
(String, L) LBuf := lp.getName();
(String, H) HBuf := lp.getName();

(XPatient, L) xp:= new XPatient();  xp.setName(LBuf:L);
(String, H) HBuf:= readTrusted...;  xp.setHIV(HBuf:H);
```

Example: Aliasing (1) Revisited

```
(LPatient, L) lp := readFile();
(String, L) LBuf := lp.getName();
(String, H) HBuf := lp.getName();

(XPatient, L) xp:= new XPatient();  xp.setName(LBuf:L);
(String, H) HBuf:= readTrusted...;  xp.setHIV(HBuf:H);

LBuf:(String,L) := HBuf:(String,H)

lp.setName(xp.getHIV():(String,H))
```

Example: Aliasing (1) Revisited

```
class LPatient L extends Object {
  (String, L) name;
  (String, L) getName() {return self.name;}
  (unit, L) setName((String, L) n) {self.name := n;} }

class XPatient L extends LPatient {
  (String, H) hiv;  //SECRET
  (String, H) getHIV() {return self.hiv;}
  (unit, L) setHIV((String, H) s) {self.hiv := s; } }
```

Example: Aliasing (2)

```
class LPatient L extends Object {
   //name, getName, setName
  (String, L) passSelf() {
     ...o.m(self)... // m has  L argument}}

 class XPatient L extends LPatient {//hiv, getHIV, setHIV}
```

Example: Aliasing (2)

```
class LPatient L extends Object {
   //name, getName, setName
  (String, L) passSelf() {
    ...o.m(self)... // m has  L argument}}

 class XPatient L extends LPatient {//hiv, getHIV, setHIV}

 class HPatient H extends XPatient {//inherits passSelf()}
```

Example: Aliasing (2)

```
class LPatient L extends Object {
  ...
  (String, L) passSelf() {...o.m(self)...}}
class XPatient L extends LPatient {...}
class HPatient H extends XPatient {//inherits passSelf()}
```

- Require: H-subclass of L-class overrides all inherited methods.

Example: Aliasing (2)

```
class LPatient L extends Object {
  ...
  (String, L) passSelf() {...o.m(self)...}}
class XPatient L extends LPatient {...}
class HPatient H extends XPatient {//inherits passSelf()}
```

- Require: H-subclass of L-class overrides all inherited methods.
- Restrictive: Why override getName?

  ```
  (String, L) getName() {return self.name;}
  ```

Example: Aliasing (2)

```
class LPatient L extends Object {
  ...
  (String, L) passSelf() {...o.m(self)...}}
class XPatient L extends LPatient {...}
class HPatient H extends XPatient {//inherits passSelf()}
```

- Require: H-subclass of L-class overrides all inherited methods.
- Restrictive: Why override getName?

  `(String, L) getName() {return self.name;}`
- Use anonymous method(?) “self” not leaked...

Example: Control Flow (Conditional)

```
class XPatient L extends LPatient {//hiv, getHIV, setHIV}

 String     leakStatus(){
   var  String     s;//level of s???
   if (self.hiv) {s := ‘‘YES’’;} else {s := ‘‘NO’’};
   return s;
}
```

Example: Control Flow (Conditional)

```
class XPatient L extends LPatient {//hiv, getHIV, setHIV}

(String, H) leakStatus(){
   var (String, H) s;//level of s???
   if (self.hiv) {s := ``YES'';} else {s := ``NO''};
   return s;
}
```

If guard is H, only H-variables and H-fields may be modified.

Example: Control Flow (Dynamic Dispatch)

```
class XPatient L extends LPatient { //hiv, ...
```

```
class YN L extends Object {(bool, L)val() {return true;}}
class Y L extends YN {(bool, L)val() {return true;}}
class N L extends YN {(bool, L)val() {return false;}}
```

Example: Control Flow (Dynamic Dispatch)

```
class XPatient L extends LPatient { //hiv, ...
 (YN, H) leak(){
   var  (YN, H) o;
   if (self.hiv) {o := new Y();} else {o := new N();}
   return o;}}


class YN L extends Object {(bool, L)val() {return true;}}
class Y L extends YN {(bool, L)val() {return true;}}
class N L extends YN {(bool, L)val() {return false;}}
```

Example: Control Flow (Dynamic Dispatch)

```
class XPatient L extends LPatient { //hiv, ...
 (YN, H) leak(){
   var  (YN, H) o;
   if (self.hiv) {o := new Y();} else {o := new N();}
   return o;}}
 xp.leak() : (YN, H);

class YN L extends Object {(bool, L)val() {return true;}}
class Y L extends YN {(bool, L)val() {return true;}}
class N L extends YN {(bool, L)val() {return false;}}
```

Example: Control Flow (Dynamic Dispatch)

```
class XPatient L extends LPatient { //hiv, ...
 (YN, H) leak(){
   var  (YN, H) o;
   if (self.hiv) {o := new Y();} else {o := new N();}
   return o;}}
 xp.leak() : (YN, H);   xp.leak().val() : (bool, ???)

class YN L extends Object {(bool, L)val() {return true;}}
class Y L extends YN {(bool, L)val() {return true;}}
class N L extends YN {(bool, L)val() {return false;}}
```

Example: Control Flow (Dynamic Dispatch)

```
class XPatient L extends LPatient { //hiv, ...
  (YN, H) leak(){...}
}
```

```
xp.leak() : (YN, H)
xp.leak().val() : (bool, H)
```

If level of receiver H, level of returned result from method call H.

Example: Dynamic Dispatch – Leaks via Heap

```
class YNh L extends Object {
  bool  v;
  bool  val() {return self.v;}
  unit setv(bool w) {self.v := w;}
  unit set() {self.setv(true);}}
```

```
class Yh L extends YNh {unit set() {self.setv(true);}}
class Nh L extends YNh {unit set() {self.setv(false);}}
```

Example: Dynamic Dispatch – Leaks via Heap

```
class YNh L extends Object {
  bool  v; //level of v???
  bool  val() {return self.v;}
  unit setv(bool w) {self.v := w;}
  unit set() {self.setv(true);}}
 x := xp.leak(); //x:(YNh, H)
 x.set();

class Yh L extends YNh {unit set() {self.setv(true);}}
class Nh L extends YNh {unit set() {self.setv(false);}}
```

Example: Dynamic Dispatch – Leaks via Heap

```
class YNh L extends Object {
  bool  v; //level of v???
  bool  val() {return self.v;}
  unit setv(bool w) {self.v := w;}
  unit set() {self.setv(true);}}
 x := xp.leak(); //x:(YNh, H)
 x.set();...x.val()...

class Yh L extends YNh {unit set() {self.setv(true);}}
class Nh L extends YNh {unit set() {self.setv(false);}}
```

Example: Dynamic Dispatch – Leaks via Heap

```
class YNh L extends Object {
  (bool, H)  v;
  (bool, H)  val() {return self.v;}
  (unit, L) setv((bool, H) w) {self.v := w;}
  (unit, L) set() {self.setv(true);}}
```

```
x := xp.leak(); x.set(); ... x.val()  :H ...
```

If level of receiver H, only H-fields may be modified in meth. call

Pointer Confinement

- L-object may be aliased by L-var, H-var
- L-class may have H-subclass

∴ Show L-Confinement:

1. L-vars, L-fields do not contain H-pointers.
2. Meaning of L-expression never H-pointer.

Pointer Confinement

- L-object may be aliased by L-var, H-var
- L-class may have H-subclass

∴ Show L-Confinement:

1. L-vars, L-fields do not contain H-pointers.
2. Meaning of L-expression never H-pointer.

- In conditionals/dyn. dispatch, assignment may be confined to H-vars, H-fields.

∴ Show H-Confinement:
Input states, output states *indistinguishable* by L.

Formalisation

Types:

$$\kappa ::= L \mid H$$

$$T ::= \texttt{unit} \mid \texttt{bool} \mid C$$

$$\tau ::= (T, \kappa)$$ //security type

Formalisation

Types:

$$\kappa ::= L \mid H$$

$$T ::= \texttt{unit} \mid \texttt{bool} \mid C$$

$$\tau ::= (T, \kappa)$$ //security type

Typing Judgements:

$$\Delta \vdash e : (T, \kappa)$$ //Δ security type context

$$\Delta \vdash S : (\texttt{com}\ \kappa_1, \kappa_2)$$

Formalisation

Types:

$$\kappa ::= \mathrm{L} \mid \mathrm{H}$$

$$T ::= \texttt{unit} \mid \texttt{bool} \mid C$$

$$\tau ::= (T, \kappa) \quad \text{//security type}$$

Typing Judgements:

$$\Delta \vdash e : (T, \kappa) \quad \text{//}\Delta \text{ security type context}$$

$$\Delta \vdash S : (\texttt{com}\ \kappa_1, \kappa_2)$$

"assign to vars $\geq \kappa_1$, update fields $\geq \kappa_2$"

Formalisation

Meanings of Typing Judgements:

$$[\![\Delta^\dagger \vdash e : T]\!]\mu\eta h, \quad [\![\Delta^\dagger \vdash S : \mathtt{com}]\!]\mu\eta h$$

$\mu \equiv$ Method Environment $\quad \eta \equiv$ Stack $\quad h \equiv$ Heap $\quad (\eta, h) \equiv$ State

$\eta \in [\![\Delta^\dagger]\!]$

Formalisation

Meanings of Typing Judgements:

$$[\![\Delta^\dagger \vdash e : T]\!]\mu\eta h, \quad [\![\Delta^\dagger \vdash S : \mathtt{com}]\!]\mu\eta h$$

$\mu \equiv$ Method Environment $\quad \eta \equiv$ Stack $\quad h \equiv$ Heap $\quad (\eta, h) \equiv$ State

$\eta \in [\![\Delta^\dagger]\!]$

Related States $(\eta, h) \sim (\eta', h')$ are *indistinguishable* by L.

$\eta \sim \eta'$, iff $\forall x \in \mathit{dom}\,\Delta$, if $(T, L) = \Delta\, x$ then $\eta\, x = \eta'\, x$;

$h \sim h'$ iff same L-locations and those have equal L-fields.

Safe Expressions

Suppose:

- $\Delta \vdash e : (T, L)$
- $(\eta, h) \sim (\eta', h')$
- μ, η, η', h, h' are L-confined
- $\mathit{safe}\ \mu$

- $[\![\Delta^\dagger \vdash e : T]\!]\mu\eta h \neq \bot \neq [\![\Delta^\dagger \vdash e : T]\!]\mu\eta' h'$

Then: $[\![\Delta^\dagger \vdash e : T]\!]\mu\eta h = [\![\Delta^\dagger \vdash e : T]\!]\mu\eta' h'$

Safe Expressions

Suppose:

- $\Delta \vdash e : (T, L)$
- $(\eta, h) \sim (\eta', h')$
- μ, η, η', h, h' are L-confined
- $\mathit{safe}\ \mu$(*i.e.*, method call in L-confined μ, η, η', h, h', $(\eta, h) \sim (\eta', h')$, yields related heaps and (if non-$\perp$) returns equal results if return type of method is L)
- $[\![\Delta^\dagger \vdash e : T]\!]\mu\eta h \neq \perp \neq [\![\Delta^\dagger \vdash e : T]\!]\mu\eta' h'$

Then: $[\![\Delta^\dagger \vdash e : T]\!]\mu\eta h = [\![\Delta^\dagger \vdash e : T]\!]\mu\eta' h'$

Safe Commands

Suppose:

- $\Delta \vdash S : (\mathtt{com}\ \kappa_1, \kappa_2)$
- μ is H-confined
- ...same as for expressions ...
- $[\![\Delta^\dagger \vdash S : \mathtt{com}]\!]\mu\eta h \neq \bot \neq [\![\Delta^\dagger \vdash S : \mathtt{com}]\!]\mu\eta' h'$.

Then *output states related*: $(\eta_0, h_0) \sim (\eta_0', h_0')$ where

$$(\eta_0, h_0) = [\![\Delta^\dagger \vdash S : \mathtt{com}]\!]\mu\eta h$$
$$(\eta_0', h_0') = [\![\Delta^\dagger \vdash S : \mathtt{com}]\!]\mu\eta' h'$$

Ongoing/Future Work

- Extension to full JavaCard
- Extension to full Java
  - Threads
  - Generics
  - ...
- Termination-insensitivity
- Inference of annotations (Pottier et. al.)
- Declassification (Halpern&O'Neill, Myers&Zdancewic)

L-confinement (ok)

- Define $LLoc = \{\ell \in Loc \mid \textit{level}\,\ell = L\}$.
- For heaps, define $ok\ h$ iff for all $\ell \in \textit{dom}\ h$ and every $f \in \textit{fields}(loctype\ \ell)$, if $\textit{stype}(f, loctype\ \ell) = (T, L)$ for some T and $h\ell f \in Loc$ then $h\ell f \in LLoc$.
- For environments, define $ok\ \Delta\,\eta$ iff for every x with $\Delta\,x = (T, L)$ for some T, if $\eta\,x \in Loc$ then $\eta\,x \in LLoc$.
- For method environments, define $ok\ \mu$ iff the following holds: for every m, C, η, h, if $ok\ h$, $ok\ \Delta\,\eta$, and $\mu C m \eta h \neq \bot$ then $\mathrm{ok}\ h_0$ and $\kappa_3 = L \wedge d \in Loc \Rightarrow d \in LLoc$,

where

$$\begin{aligned}
\textit{smtype}(m, C) &= (\overline{T}, \overline{\kappa}) \xrightarrow{\kappa_2} (T, \kappa_3) \\
\textit{pars}(m, C) &= (\overline{x} : (\overline{T}, \overline{\kappa})) \\
\Delta &= \overline{x} : (\overline{T}, \overline{\kappa}), \mathsf{self} : (C, \textit{level}\, C) \\
(d, h_0) &= \mu C m \eta h
\end{aligned}$$

Type Rules

$$\frac{C = \Gamma\,\mathsf{self} \qquad T f \in \mathit{dfields}\, C \qquad \Gamma \vdash e_1 : C \qquad \Gamma \vdash e_2 : U \quad U \leq T}{\Gamma \vdash e_1.f := e_2 : \mathtt{com}}$$

Type Rules

$$\frac{C = \Gamma\,\mathsf{self} \qquad T\,f \in \mathit{dfields}\,C \qquad \Gamma \vdash e_1 : C \qquad \Gamma \vdash e_2 : U \quad U \leq T}{\Gamma \vdash e_1.f := e_2 : \mathtt{com}}$$

$$\frac{\mathit{mtype}(m, D) = \overline{T} \to T \qquad \Gamma \vdash e : D \qquad \Gamma \vdash \overline{e} : \overline{U} \qquad \overline{U} \leq \overline{T}}{\Gamma \vdash e.m(\overline{e}) : \mathtt{com}}$$

Security Type Rules

$$\frac{x \neq \mathsf{self} \quad T_2 \leq T_1 \quad \kappa_2 \leq \kappa_1 \quad \kappa_3 \leq \kappa_1 \qquad \Delta, x : (T_1, \kappa_1) \vdash e : (T_2, \kappa_2)}{\Delta, x : (T_1, \kappa_1) \vdash x := e : (\mathtt{com}\ \kappa_3, \kappa_4)}$$

Security Type Rules

$$\frac{x \neq \mathsf{self} \quad T_2 \leq T_1 \quad \kappa_2 \leq \kappa_1 \quad \kappa_3 \leq \kappa_1 \qquad \Delta, x : (T_1, \kappa_1) \vdash e : (T_2, \kappa_2)}{\Delta, x : (T_1, \kappa_1) \vdash x := e : (\mathtt{com}\ \kappa_3, \kappa_4)}$$

$$\frac{(T, \kappa_2) f \in \mathit{sdfields}\, C \qquad \Delta \vdash e_1 : (C, \kappa_1) \quad \Delta \vdash e_2 : (U, \kappa_3) \qquad U \leq T \quad \kappa_1 \sqcup \kappa_3 \sqcup \kappa_5 \leq \kappa_2}{\Delta \vdash e_1.f := e_2 : (\mathtt{com}\ \kappa_4, \kappa_5)}$$

$$\frac{\begin{array}{l} \textit{smtype}(m, D) = (\overline{T}, \overline{\kappa}) \xrightarrow{\kappa_3} (T, \kappa_2) \\ \Delta \vdash e : (D, \kappa_4) \qquad \Delta \vdash \overline{e} : (\overline{U}, \overline{\kappa_5}) \\ \overline{U} \leq \overline{T} \quad \overline{\kappa_5} \leq \overline{\kappa} \quad \kappa_4 \sqcup \kappa_7 \leq \kappa_3 \end{array}}{\Delta \vdash e.m(\overline{e}) : (\mathtt{com}\ \kappa_6, \kappa_7)}$$