IFlow – Model driven development of information flow secure systems

Wolfgang Reif, Kuzman Katkalov, Marian Borek, Kurt Stenzel
Where does your information go?

- **Travel Agency** *never learns* the user’s credit card data
- **Airline** only learns the user’s credit card data *after user confirmation*
How much information are you sharing?

- Health Insurance Service never learns the user’s location
- Distance Tracker properly anonymizes the user’s location data as distance

Prof. Dr. W. Reif, reif@isse.de
Overview

- IFlow: a model driven approach for developing distributed apps with secure information flow

- Software/security co-design in distributed apps
  - Modeling
  - Automatic check and verification

- Special aspect: limited release of information
  - Modeling
  - Verification
IFlow: a model-driven approach to IFC

Modeling language

- principle: security by design
- graphical, domain specific language for IFC
- distributed apps & services
- with tool support
- and code generation

Formal foundation

- formal semantics (ASMs & predicate logic)
- intransitive noninterference (extension of Rushby/Meyden)
- declassification ("what", "where", "when")
IFlow: a model-driven approach to IFC

Prof. Dr. W. Reif, reif@isse.de

Result:
runnable application with guaranteed information flow properties
A hybrid approach to provide guarantees

Information flow:
Automatic code check

Travel Agency *never learns* the user’s credit card data

Health Insurance Service *never learns* the user’s location

Information release:
Formal verification

Airline only learns the user’s credit card data *after user confirmation*

Distance Tracker *properly anonymizes* the user’s location

Prof. Dr. W. Reif, reif@isse.de
Overview

• IFlow: a model driven approach for developing distributed apps with secure information flow

• Software/security co-design in distributed apps
  – Modeling
  – Automatic check and verification

• Special aspect: limited release of information
  – Modeling
  – Verification
TravelPlanner: components

Application components: user interface, mobile apps and web services
TravelPlanner: data types

Data types and message classes
TravelPlanner: behaviour

1. GetInput(ClientUser, TravelAgency, Airline)
2. GetFlightOffers(requestData = input) (ClientUser, TravelAgency, Airline)
3. GetFlightOffers(requestData) (User, TravelAgency, Airline)
4. GetFlightOffers(requestData) (User, TravelAgency, Airline)
5. filteredFlightOffers = filterFlightOffers(requestData, flightOffers) (User, TravelAgency, Airline)
6. filterFlightOffers(flightOffers) (User, TravelAgency, Airline)
7. ReFilterFlightOffers(flightOffers) (User, TravelAgency, Airline)
8. GetSingleSelection(flightOffers) (User, Airline)
9. BookSelected(flightOffer = singleSelection) (User, Airline)
10. ReleaseCCD(airline = flightOffer.airline) (User)
11. ConfirmRelease(ccd, airline) (User)
12. GetDeclassifiedCCD() (User)
13. ccd_decl := declassifyCCD(ccd) (User, Airline)
14. DeclassifiedCCD(ccd_decl) (User, Airline)
15. BookFlightOffer(id := flightOffer.id, ccd_decl) (User, Airline)
16. processBooking(id, ccd_decl) (User, Airline)
17. GetCommission() (User, Airline)
18. GetPayCommission() (User, TravelAgency, Airline)
19. OkConfirmBooking() (User, TravelAgency, Airline)
20. Confirm() (User, TravelAgency, Airline)

confirmation of credit card release
Travel Planner: security

• **Subjects** (user, apps & web services): $S$
  
  $S_{TravelPlanner} = \{User, Airline, TravelAgency\}$

• **Security domains**: $d \subseteq S$, $d \subseteq S \times S$
  
  $\{User\}$, $\{User, Airline\}$, $\{User, Airline, TravelAgency\}$, $\{User\} \rightarrow \{User, Airline\}$

• **Security policy**:
Assigning security domains to data & messages:

- {User} -> {User, Airline}
- {User, Airline} -> {User, TravelAgency, Airline}

13: ccd_decl := declassify CCD(ccd)
{User -> User, Airline}

17: PayCommission()
{User, TravelAgency, Airline}
TravelPlanner: behaviour

1. GetInput(RequestData)
   (User, TravelAgency, Airline)

2. GetFlightOffers(requestData = input)
   (User, TravelAgency, Airline)

3. GetFlightOffers(requestData)
   (User, TravelAgency, Airline)

4. GetFlightOffers(requestData)
   (User, TravelAgency, Airline)

5. GetFlightOffers(filterOffers = filteredFlightOffers)
   (User, TravelAgency, Airline)

6. GetFlightOffers(filterOffers = filteredFlightOffers)
   (User, TravelAgency, Airline)

7. ReFlightOffers(filterOffers)
   (User, TravelAgency, Airline)

8. GetSingleSelection(filterOffers)
   (User, Airline)

9. BookSelected(flightOffer = singleSelection)
   (User, Airline)

10. ReleaseCCD(airline = flightOffer.airline)
    (User)

11. ConfirmRelease(ccd, airline)
    (User)

12. GetDeclassifiedCCD()
    (User)

13. ccd_dec := declassifyCCD(ccd)
    (User, User, Airline)

14. DeclassifiedCCD(ccd_dec)
    (User, User, Airline)

15. BookFlightOffer(id := flightOffer.id, ccd_dec)
    (User, Airline)

16. processBooking(id, ccd_dec)
    (User, Airline)

17. PayComission()
    (User, Airline)

18. OkPayCommission()
    (User, TravelAgency, Airline)

19. OKConfirmBooking()
    (User, TravelAgency, Airline)

20. Confirm()
    (User, TravelAgency, Airline)

confirmation of credit card release

Prof. Dr. W. Reif, reif@isse.de
Travel Planner: **security (cont.)**

**Noninterference theorem**  
(extension of Rushby/van der Meyden):

---

**Security policy satisfied, if**

RM1: \( s_1 \approx_d s_2 \rightarrow \text{output}(s_1, d) = \text{output}(s_2, d) \)  
RM2: \( s_1 \approx_{\text{dom}(a)} s_2 \land s_1(l) = s_2(l) \land l \in \text{alter}(\text{dom}(a)) \rightarrow \text{step}(s_1, a)(l) = \text{step}(s_2, a)(l) \)  
RM3: \( \text{step}(s, a)(l) \neq s(l) \rightarrow l \in \text{alter}(\text{dom}(a)) \)  
AOI: \( \text{alter}(d_1) \cap \text{observe}(d_2) \neq \emptyset \rightarrow d_1 \sim d_2 \)

(true by construction) ✔
(true by construction) ✔
(true by construction) ✔
(prove)
TravelPlanner: security properties

- Components (apps/services) and their attributes can be **sources** and **sinks** of information.
  - Information flow is either
    - **disallowed** («noFlow»), or
    - **allowed** («allowedFlow»)
      ... via a specific method («via»)
      ... after **user confirmation** ([confirmation])

- Properties can be
  - **automatically checked** on the code level, or
  - **verified interactively** using the formal model.
TravelPlanner: **automatic IF check**

1. Generate a code skeleton as a **monolithic Java application**

2. Generate and run **information flow check** with JOANA (PDG-based IFC)

3. Distribute application as **Android apps** and **Java web services**

Prof. Dr. W. Reif, reif@isse.de
TravelPlanner: verification

**Proof obligations:**

1. Security policy implies property

2. Application model satisfies security policy (noninterference theorem)

3. `declassifyCCD` is only called after confirmation

- \( \neg \text{domain}(\text{ccd}) \Rightarrow \text{domain}(\text{Airline}) \land \text{domain}(\text{ccd}) \Rightarrow \text{domain}(\text{declassifyCCD}) \land \text{domain}(\text{declassifyCCD}) \Rightarrow \text{domain}(\text{Airline}) \)
Overview

• IFlow: a model driven approach for developing distributed apps with secure information flow

• Software/security co-design in distributed apps
  – Modeling
  – Automatic check and verification

• Special aspect: limited release of information
  – Modeling
  – Verification

Prof. Dr. W. Reif, reif@isse.de
Distance Tracker: **property**

- **Tracker Service** *never learns* the user’s location
- **Distance Tracker** *properly anonymizes* the user’s location data as distance
What do we prove?

• define equivalence class of all inputs that map to the same output
• reason about size of the classes

„For every valid output there are enough (e) different inputs with mutually disjoint positions“:

\[ \forall x, d. \text{calcDist}(x) = d \rightarrow \exists \text{a_set}. \# \text{a_set} \geq e \land \text{disjoint}(\text{a_set}) \land \forall y. y \in \text{a_set} \rightarrow \text{calcDist}(y) = d \]
calcDist(act : Activity) : Distance

Activity
- nickname : String
- positions 1..* {ordered}

GPSPos
- latitude : double
- longitude : double

Distance
- distance : Integer
- nickname : String
- appInfo : String

\[
\text{calcDist} = \begin{cases}
\text{dist} = 0; & \text{if (p1.latitude - p2.latitude)^2 + (p1.longitude - p2.longitude)^2 = 0} \\
\text{dist} = \text{Math.sqrt}((\text{Math.cos}(\text{Math.toRadians}(\text{lat1}) - \text{Math.toRadians}(\text{lat2})) * \text{Math.cos}(\text{Math.toRadians}(\text{lon1}) - \text{Math.toRadians}(\text{lon2}))) \text{sqrt}(\text{a}^2 + \text{b}^2 + \text{c}^2); & \text{otherwise}
\end{cases}
\]

where:
- \(\text{a} = \text{Math.toRadians}(\text{lat1}) - \text{Math.toRadians}(\text{lat2})\)
- \(\text{b} = \text{Math.toRadians}(\text{lon1}) - \text{Math.toRadians}(\text{lon2})\)
- \(\text{c} = 2 \times \text{Math.sqrt}(\text{Math.sqrt}(\text{a}^2 + \text{b}^2) - 1)\)
- \(\text{R} = 6371; \text{Radius of the earth}\)
- \(\text{d} = \text{Math.sqrt}(\text{a}^2 + \text{b}^2 + \text{c}^2); \text{distance in meters}\)
IFlow integrates
- a model driven approach (ModelFlow)
- formally verified information flow properties
- automatic information flow control