UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Securing Cross-App Interactions in IoT Platforms

Musard Balliu, Massimo Merro and Michele Pasqua ✉

June 27, 2019

✉ michele.pasqua@univr.it

32nd IEEE Computer Security Foundations Symposium (CSF) - Hoboken, USA

Introduction

Scenario

Platforms for IoT Apps

Scenario

Platforms for IoT Apps

Physical devices

Scenario

Platforms for IoT Apps

Physical devices

Digital services

Scenario

Platforms for IoT Apps

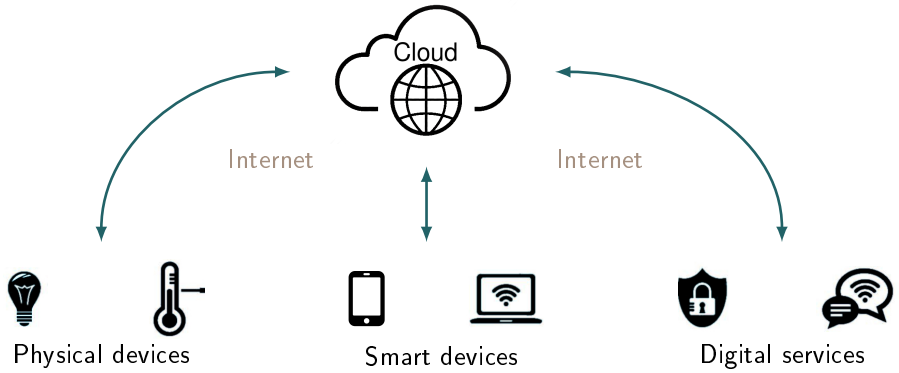
Physical devices

Smart devices

Digital services

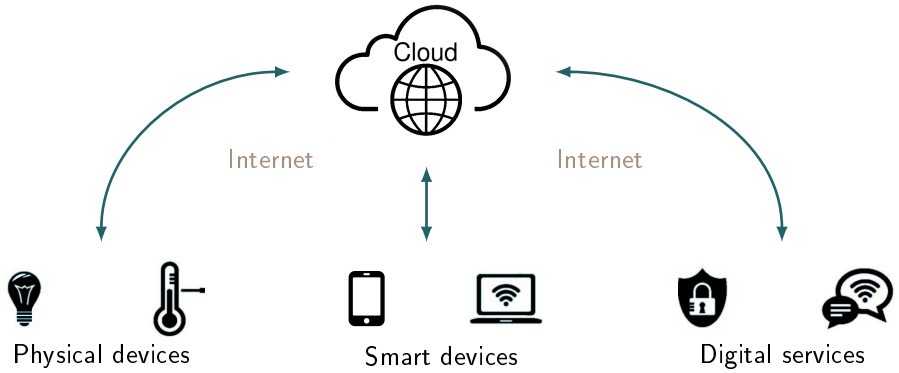
Scenario

Platforms for IoT Apps



Scenario

Platforms for IoT Apps



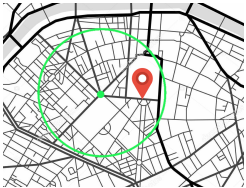
IoT apps platforms: IFTTT, Stringify, Microsoft Flow, etc

How it works

Trigger/action paradigm

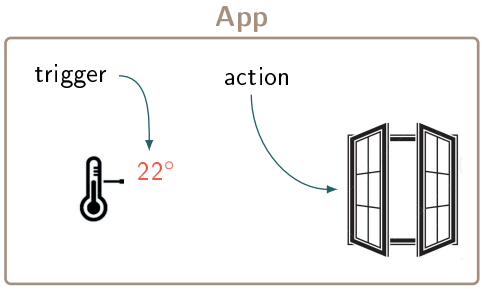
App

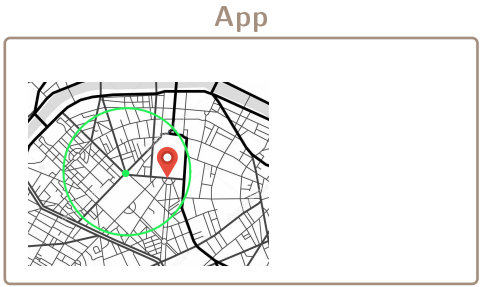
App



How it works

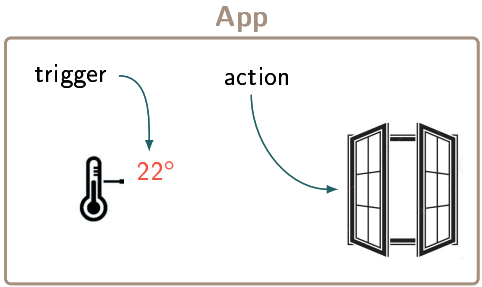
Trigger/action paradigm

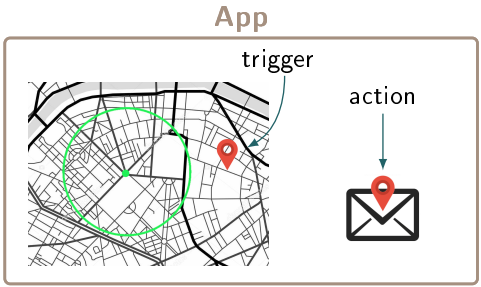




How it works

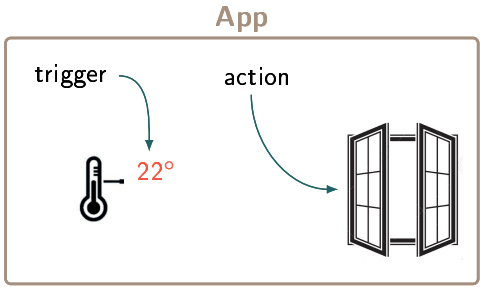
Trigger/action paradigm

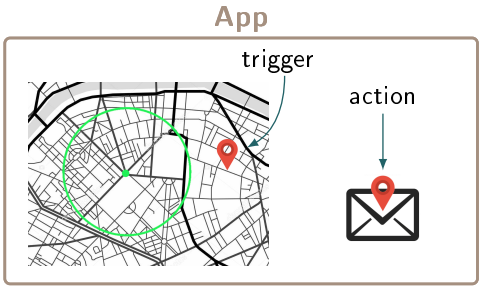




How it works

Trigger/action paradigm





- Filter code before actions execution
- Third-parties apps development (malicious apps)

Vulnerabilities: Safety and Security

- Safety: unintended cross-app interactions

Vulnerabilities: Safety and Security

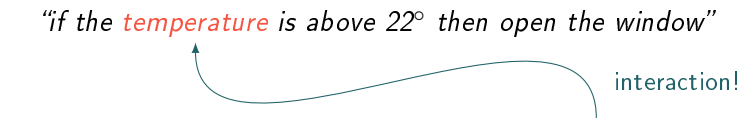
- Safety: unintended cross-app interactions

"if the temperature is above 22° then open the window"

"if I leave my work location then turn on the heater at home"

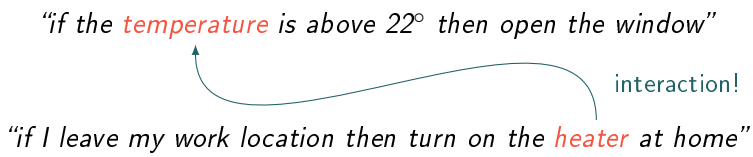
Vulnerabilities: Safety and Security

- Safety: unintended cross-app interactions



Vulnerabilities: Safety and Security

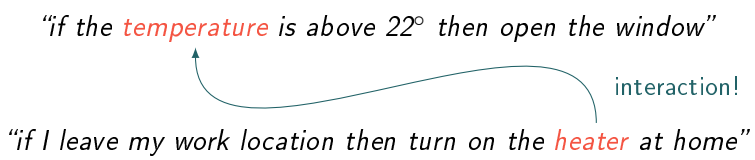
- Safety: unintended cross-app interactions



- Security: confidentiality violations (information flows)

Vulnerabilities: Safety and Security

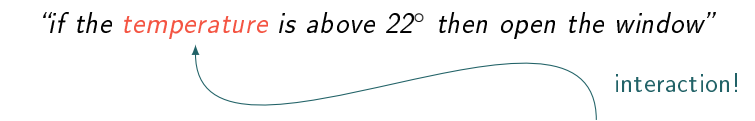
- Safety: unintended cross-app interactions



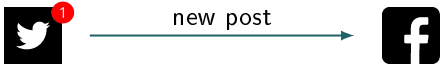
- Security: confidentiality violations (information flows)

Vulnerabilities: Safety and Security

- Safety: unintended cross-app interactions

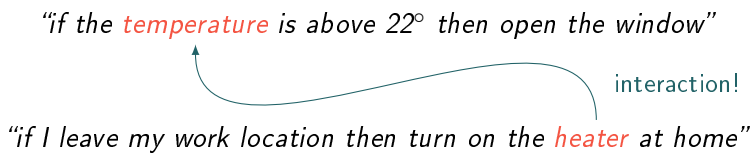


- Security: confidentiality violations (information flows)



Vulnerabilities: Safety and Security

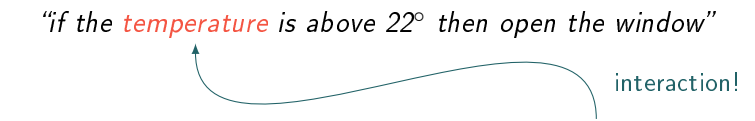
- Safety: unintended cross-app interactions



- Security: confidentiality violations (information flows)

Vulnerabilities: Safety and Security

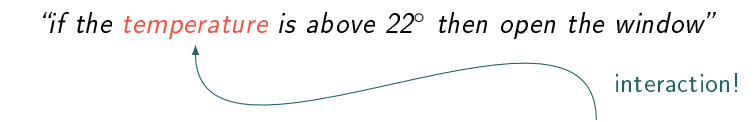
- Safety: unintended cross-app interactions



- Security: confidentiality violations (information flows)

Vulnerabilities: Safety and Security

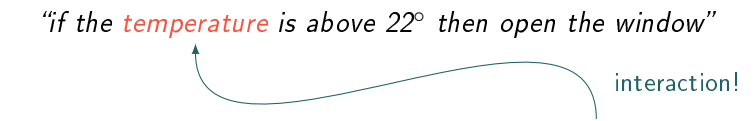
- Safety: unintended cross-app interactions



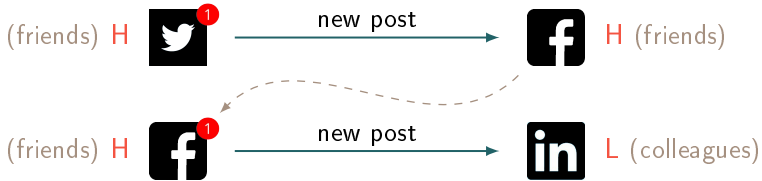
- Security: confidentiality violations (information flows)

Vulnerabilities: Safety and Security

- Safety: unintended cross-app interactions



- Security: confidentiality violations (information flows)



Goal: Securing IoT Apps

A formal framework for proving safety and security of IoT platforms

Goal: Securing IoT Apps

A formal framework for proving safety and security of IoT platforms

1. Formal model: formal language for IoT platforms
   - process calculus approach

Goal: Securing IoT Apps

A formal framework for proving safety and security of IoT platforms

1. Formal model: formal language for IoT platforms
   - process calculus approach
2. Semantic conditions: safety and security requirements definition
   - bisimulation-based algebraic laws

Goal: Securing IoT Apps

A formal framework for proving safety and security of IoT platforms

1. Formal model: formal language for IoT platforms
   - process calculus approach
2. Semantic conditions: safety and security requirements definition
   - bisimulation-based algebraic laws
3. Enforcement mechanisms: sufficient syntactic conditions
   - safety: syntactic constraints for triggers and actions
   - security: flow-sensitive type system

Goal: Securing IoT Apps

A formal framework for proving safety and security of IoT platforms

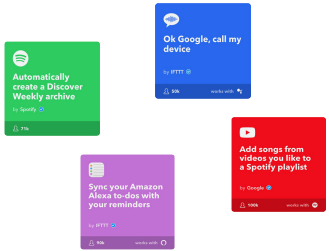
1. Formal model: formal language for IoT platforms
   - process calculus approach
2. Semantic conditions: safety and security requirements definition
   - bisimulation-based algebraic laws
3. Enforcement mechanisms: sufficient syntactic conditions
   - safety: syntactic constraints for triggers and actions
   - security: flow-sensitive type system

Future enforcement mechanisms can be proven sound w.r.t. our semantic conditions

The Calculus

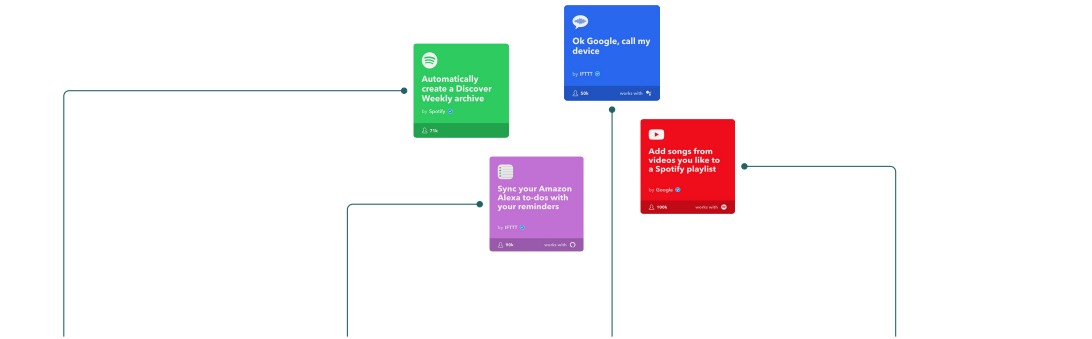
A Calculus for IoT Apps: Features

Collection of apps belonging to the same user



A Calculus for IoT Apps: Features

Collection of apps belonging to the same user



A Calculus for IoT Apps: Features

Collection of apps belonging to the same user

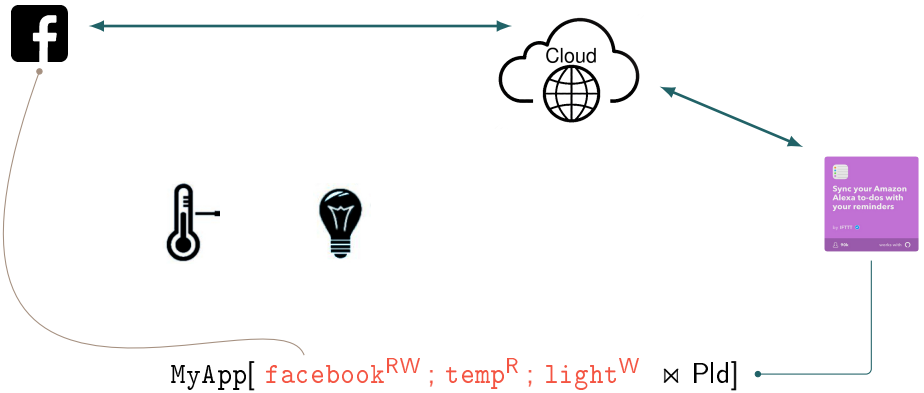
$$\texttt{Discover}[\mathrm{Dcl}_{\mathrm{D}} \bowtie \mathrm{Pld}_{\mathrm{D}}] \parallel \texttt{Alexa}[\mathrm{Dcl}_{\mathrm{A}} \bowtie \mathrm{Pld}_{\mathrm{A}}] \parallel \texttt{CallDev}[\mathrm{Dcl}_{\mathrm{C}} \bowtie \mathrm{Pld}_{\mathrm{C}}] \parallel \texttt{Spotify}[\mathrm{Dcl}_{\mathrm{S}} \bowtie \mathrm{Pld}_{\mathrm{S}}]$$

A Calculus for IoT Apps: Features (contd.)

Physical and Logical resources are cloud services

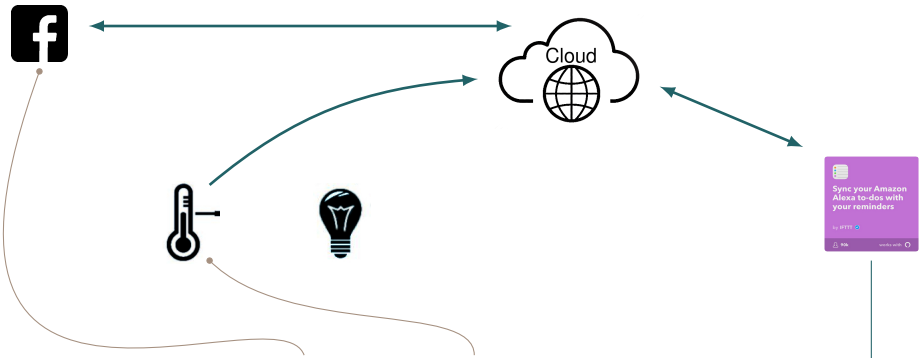
A Calculus for IoT Apps: Features (contd.)

Physical and Logical resources are cloud services



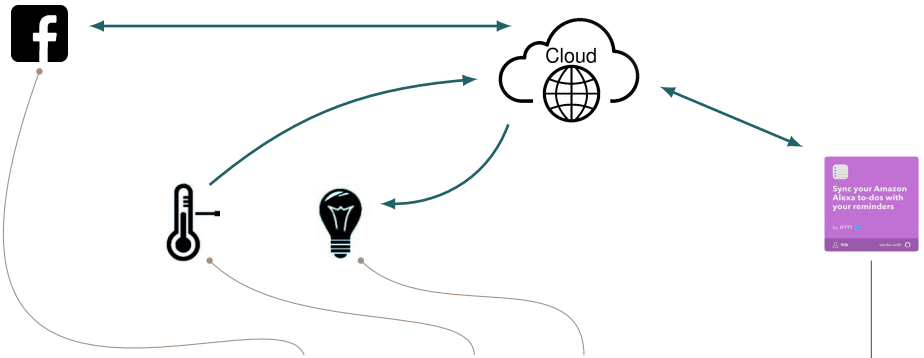
A Calculus for IoT Apps: Features (contd.)

Physical and Logical resources are cloud services



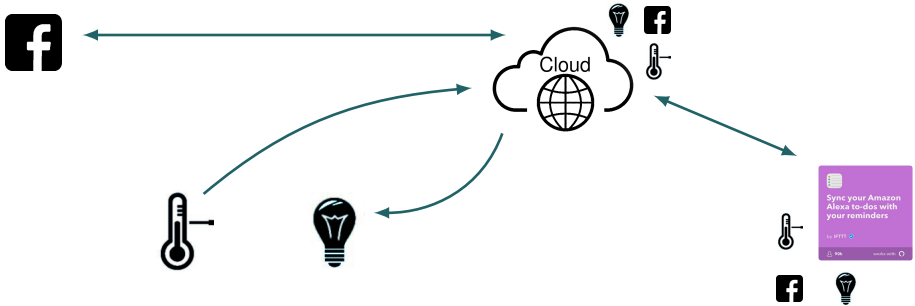
A Calculus for IoT Apps: Features (contd.)

Physical and Logical resources are cloud services



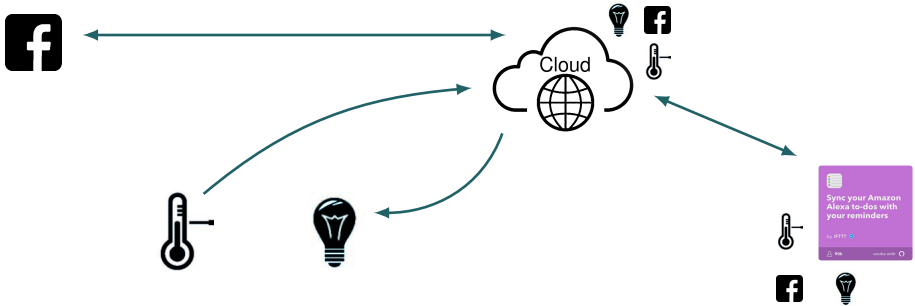
A Calculus for IoT Apps: Features (contd.)

Cloud and local view of services



A Calculus for IoT Apps: Features (contd.)

Cloud and local view of services



Apps retain a local view of cloud services

A Calculus for IoT Apps: Features (contd.)

Triggers

local services differ from the cloud ones
activation condition check

Global look-up

retrieves the values of cloud services

Isolated execution

computations affect only local services

Actions

update cloud services with local info

SmokeAlarm

$$\mathsf{fix}\ \mathbb{X} \bullet \mathtt{listen}(\mathtt{smoke})\ ;\ \mathtt{smoke} \leftarrow \mathtt{read}(\mathtt{smoke})\ ;\ \mathsf{if}\ (\mathtt{smoke} = \mathit{yes})\ \mathsf{then}\ \{\ \mathtt{alarm} \leftarrow \mathit{On}\ ;\ \mathtt{lights} \leftarrow \mathit{On}\ ;\ \mathtt{update}(\mathtt{alarm}, \mathtt{lights})\ \}\ ;\ \mathbb{X}$$

A Calculus for IoT Apps: Features (contd.)

Triggers

local services differ from the cloud ones
activation condition check

Global look-up

retrieves the values of cloud services

Isolated execution

computations affect only local services

Actions

update cloud services with local info

SmokeAlarm

```
fix 𝕏 • listen(smoke) ;
    smoke ← read(smoke) ;
    if (smoke = yes) then {
        alarm ← On ; lights ← On ;
        update(alarm, lights)
    } ; 𝕏
```

A Calculus for IoT Apps: Features (contd.)

Triggers

local services differ from the cloud ones
activation condition check

Global look-up

retrieves the values of cloud services

Isolated execution

computations affect only local services

Actions

update cloud services with local info

SmokeAlarm

$$\mathtt{fix}\ \mathbb{X} \bullet \mathtt{listen}(\mathtt{smoke})\ ;\ \mathtt{smoke} \leftarrow \mathtt{read}(\mathtt{smoke})\ ;\ \mathtt{if}\ (\mathtt{smoke} = \mathit{yes})\ \mathtt{then}\ \{\ \mathtt{alarm} \leftarrow \mathit{On}\ ;\ \mathtt{lights} \leftarrow \mathit{On}\ ;\ \mathtt{update}(\mathtt{alarm}, \mathtt{lights})\ \}\ ;\ \mathbb{X}$$

A Calculus for IoT Apps: Features (contd.)

Triggers

local services differ from the cloud ones
activation condition check

Global look-up

retrieves the values of cloud services

Isolated execution

computations affect only local services

Actions

update cloud services with local info

SmokeAlarm

$$\mathsf{fix}\ \mathbb{X} \bullet \mathsf{listen}(\mathtt{smoke})\ ;\ \mathtt{smoke} \leftarrow \mathsf{read}(\mathtt{smoke})\ ;\ \mathsf{if}\ (\mathtt{smoke} = \mathit{yes})\ \mathsf{then}\ \{\ \mathtt{alarm} \leftarrow \mathit{On}\ ;\ \mathtt{lights} \leftarrow \mathit{On}\ ;\ \mathsf{update}(\mathtt{alarm}, \mathtt{lights})\ \}\ ;\ \mathbb{X}$$

A Calculus for IoT Apps: Features (contd.)

Triggers

local services differ from the cloud ones
activation condition check

Global look-up

retrieves the values of cloud services

Isolated execution

computations affect only local services

Actions

update cloud services with local info

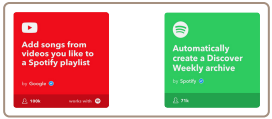
SmokeAlarm

$$\mathtt{fix}\ \mathbb{X} \bullet \mathtt{listen}(\mathtt{smoke})\ ;\ \mathtt{smoke} \leftarrow \mathtt{read}(\mathtt{smoke})\ ;\ \mathtt{if}\ (\mathtt{smoke} = \mathit{yes})\ \mathtt{then}\ \{\ \mathtt{alarm} \leftarrow \mathit{On}\ ;\ \mathtt{lights} \leftarrow \mathit{On}\ ;\ \mathtt{update}(\mathtt{alarm}, \mathtt{lights})\ \}\ ;\ \mathbb{X}$$

Safety Condition

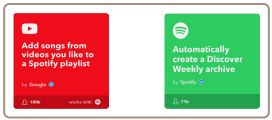
Safety: Cross-App Interactions

app1 noninteracting with app2:



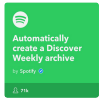
Safety: Cross-App Interactions

app1 noninteracting with app2:



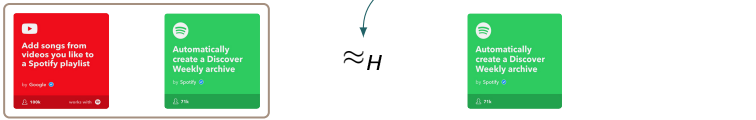
$\approx_H$

Hiding bisimulation



Safety: Cross-App Interactions

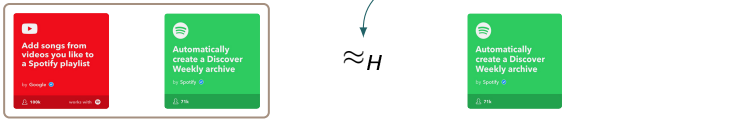
app1 noninteracting with app2:



- $\approx_H$ hides the observables contained in H

Safety: Cross-App Interactions

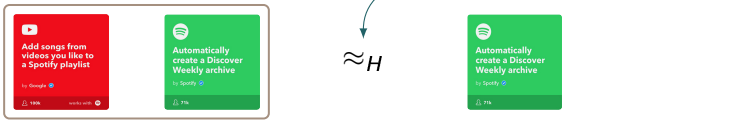
app1 noninteracting with app2:



- $\approx_H$ hides the observables contained in H
- Observables are modifications of the cloud

Safety: Cross-App Interactions

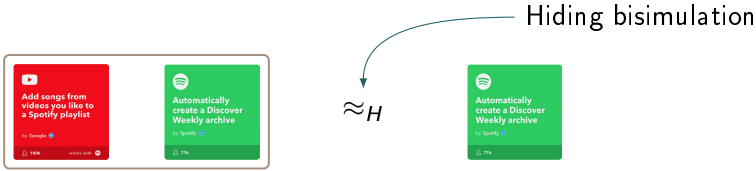
app1 noninteracting with app2:



- $\approx_H$ hides the observables contained in H
- Observables are modifications of the cloud
- We need to hide the updates (writes) made by app1

Safety: Cross-App Interactions

app1 noninteracting with app2:

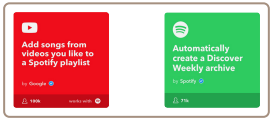


- $\approx_H$ hides the observables contained in H
- Observables are modifications of the cloud
- We need to hide the updates (writes) made by app1

$$\text{app1} \parallel \text{app2} \approx_{H_{\text{app1}}} \text{app2}$$

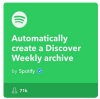
Safety: Cross-App Interactions

app1 noninteracting with app2:



Hiding bisimulation

$\approx_H$



- $\approx_H$ hides the observables contained in H
- Observables are modifications of the cloud
- We need to hide the updates (writes) made by app1

$$S \parallel R \approx_{H_S} R$$

Enforcement: Syntactic Constraints for Triggers and Actions

Example of a simple syntactic enforcement mechanism

app1 noninteracting with app2:

- apps do not update common services $\quad \text{actions}(\texttt{app1}) \cap \text{actions}(\texttt{app2}) = \varnothing$
- app1 does not trigger app2 $\quad \text{actions}(\texttt{app1}) \cap \text{triggers}(\texttt{app2}) = \varnothing$

Enforcement: Syntactic Constraints for Triggers and Actions

Example of a simple syntactic enforcement mechanism

app1 noninteracting with app2:

- apps do not update common services $\quad \text{actions}(\texttt{app1}) \cap \text{actions}(\texttt{app2}) = \varnothing$
- app1 does not trigger app2 $\quad \text{actions}(\texttt{app1}) \cap \text{triggers}(\texttt{app2}) = \varnothing$

Soundness: the syntactic enforcement implies the semantic safety condition

Enforcement: Syntactic Constraints for Triggers and Actions (contd.)

SmokeAlarm

```
fix 𝕏 • listen(smoke) ;
    smoke ← read(smoke) ;
    if (smoke = yes) then {
        alarm ← On ; lights ← On ;
        update(alarm, lights)
    } ; 𝕏
```

Sprinks

```
fix 𝕏 • listen(heat) ;
    heat ← read(heat) ;
    if (heat ≥ 45) then {
        waterV ← Open ;
        update(waterV)
    } ; 𝕏
```

Enforcement: Syntactic Constraints for Triggers and Actions (contd.)

SmokeAlarm

$$
\begin{array}{l}
\mathsf{fix}\ \mathbb{X} \bullet \mathtt{listen(smoke)}\ ; \\
\quad \mathtt{smoke} \leftarrow \mathtt{read(smoke)}\ ; \\
\quad \mathtt{if}\ \ (\mathtt{smoke} = \mathit{yes})\ \ \mathtt{then}\ \ \{ \\
\qquad \mathtt{alarm} \leftarrow \mathit{On}\ ;\ \mathtt{lights} \leftarrow \mathit{On}\ ; \\
\qquad \mathtt{update(alarm, lights)} \\
\quad \}\ ;\ \mathbb{X}
\end{array}
$$

Sprinks

$$
\begin{array}{l}
\mathsf{fix}\ \mathbb{X} \bullet \mathtt{listen(heat)}\ ; \\
\quad \mathtt{heat} \leftarrow \mathtt{read(heat)}\ ; \\
\quad \mathtt{if}\ \ (\mathtt{heat} \geq 45)\ \ \mathtt{then}\ \ \{ \\
\qquad \mathtt{waterV} \leftarrow \mathit{Open}\ ; \\
\qquad \mathtt{update(waterV)} \\
\quad \}\ ;\ \mathbb{X}
\end{array}
$$

Semantic safety: $\mathtt{Sprinks} \parallel \mathtt{SmokeAlarm} \approx_{H_{\mathtt{Sprinks}}} \mathtt{SmokeAlarm}$

Enforcement: Syntactic Constraints for Triggers and Actions (contd.)

SmokeAlarm

```
fix 𝕏 • listen(smoke) ;
      smoke ← read(smoke) ;
      if (smoke = yes) then {
           alarm ← On ; lights ← On ;
           update(alarm, lights)
      } ; 𝕏
```

Sprinks

```
fix 𝕏 • listen(heat) ;
      heat ← read(heat) ;
      if (heat ≥ 45) then {
           waterV ← Open ;
           update(waterV)
      } ; 𝕏
```

Semantic safety: $\texttt{Sprinks} \parallel \texttt{SmokeAlarm} \approx_{H_{\texttt{Sprinks}}} \texttt{SmokeAlarm}$

We can state safety syntactically:

- apps do not update common services $\{\texttt{waterV}\} \cap \{\texttt{alarm}, \texttt{lights}\} = \varnothing$
- Sprinks does not trigger SmokeAlarm $\{\texttt{waterV}\} \cap \{\texttt{smoke}\} = \varnothing$

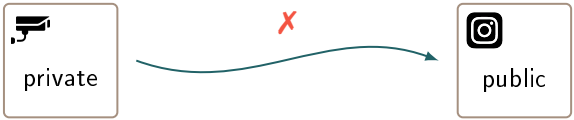
Security Condition

Security: Information Flows

Services with different security clearance

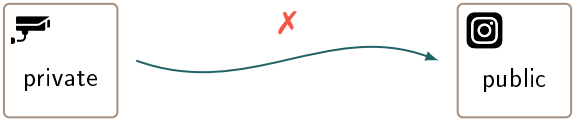
Security: Information Flows

Services with different security clearance



Security: Information Flows

Services with different security clearance



Security policy:

- lattice of security levels $\langle \mathrm{SL}, \preccurlyeq \rangle$
- security levels assignment Σ to services
- σ-equivalence $\equiv_\sigma$: stores agree on services with security levels $\preccurlyeq \sigma$

Security: Information Flows (contd.)

Noninterference:

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

Security: Information Flows (contd.)

Noninterference:

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx_{H_\sigma} \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

hide updates greater than σ

Security: Information Flows (contd.)

Noninterference:

ignore "termination" leakage — [Demange and Sands, ESOP 2009]

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx^{\mathrm{ti}}_{H_\sigma} \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

hide updates greater than σ

Security: Information Flows (contd.)

Noninterference:

ignore "termination" leakage — [Demange and Sands, ESOP 2009]

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx^{\mathrm{ti}}_{H_\sigma} \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

ignore "presence" leakage

hide updates greater than σ

Security: Information Flows (contd.)

Noninterference:

ignore "termination" leakage — [Demange and Sands, ESOP 2009]

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx^{\mathrm{ti}}_{H_\sigma} \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

ignore "presence" leakage

hide updates greater than σ

`Tw2Fb`

fix $\mathbb{X}$ • listen(`tw`) ; `tw` ← read(`tw`) ;
`fb` ← *tw* ; update(`fb`) ; $\mathbb{X}$

`Fb2Ld`

fix $\mathbb{X}$ • listen(`fb`) ; `fb` ← read(`fb`) ;
`ld` ← *fb* ; update(`ld`) ; $\mathbb{X}$

Security: Information Flows (contd.)

Noninterference:

ignore "termination" leakage — [Demange and Sands, ESOP 2009]

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx^{\mathrm{ti}}_{H_\sigma} \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

ignore "presence" leakage

hide updates greater than σ

Tw2Fb

fix $\mathbb{X}$ • listen(tw) ; tw ← read(tw) ;
fb ← *tw* ; update(fb) ; $\mathbb{X}$

Fb2Ld

fix $\mathbb{X}$ • listen(fb) ; fb ← read(fb) ;
ld ← *fb* ; update(ld) ; $\mathbb{X}$

$\Sigma(\mathtt{tw}) = \Sigma(\mathtt{fb}) = \mathsf{H}$ and $\Sigma(\mathtt{ld}) = \mathsf{L}$

$$\langle \mathfrak{G}, \mathfrak{L}_\perp \rangle \triangleright \mathtt{Tw2Fb} \parallel \mathtt{Fb2Ld} \not\approx^{\mathrm{ti}}_{H_\mathsf{L}} \langle \mathfrak{G}', \mathfrak{L}_\perp \rangle \triangleright \mathtt{Tw2Fb} \parallel \mathtt{Fb2Ld}$$

Security: Information Flows (contd.)

Noninterference:

ignore "termination" leakage — [Demange and Sands, ESOP 2009]

$$\forall \mathfrak{G}, \mathfrak{G}' \in \mathbb{S}.\ \mathfrak{G} \equiv_\sigma \mathfrak{G}' \Rightarrow \langle \mathfrak{G}, \mathfrak{L} \rangle \triangleright \mathsf{S} \approx^{\mathsf{ti}}_{H_\sigma} \langle \mathfrak{G}', \mathfrak{L} \rangle \triangleright \mathsf{S}$$

ignore "presence" leakage

hide updates greater than σ

Tw2Fb

fix $\mathbb{X}$ • listen(tw) ; tw ← read(tw) ;
fb ← *tw* ; update(fb) ; $\mathbb{X}$

Fb2Ld

fix $\mathbb{X}$ • listen(fb) ; fb ← read(fb) ;
ld ← *fb* ; update(ld) ; $\mathbb{X}$

$\Sigma(\mathtt{tw}) = \Sigma(\mathtt{fb}) = \mathsf{H}$ and $\Sigma(\mathtt{ld}) = \mathsf{H}$

$$\langle \mathfrak{G}, \mathfrak{L}_\perp \rangle \triangleright \mathtt{Tw2Fb} \parallel \mathtt{Fb2Ld} \approx^{\mathsf{ti}}_{H_\mathsf{L}} \langle \mathfrak{G}', \mathfrak{L}_\perp \rangle \triangleright \mathtt{Tw2Fb} \parallel \mathtt{Fb2Ld}$$

Enforcement: Security Type System

Enforcement

- flow-sensitive security type system [Hunt and Sands, POPL 2006]
- input/output check-points

Enforcement: Security Type System

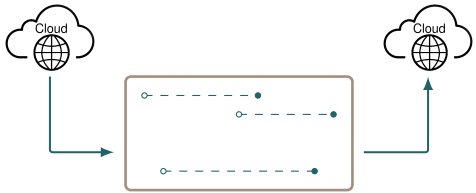
Enforcement

- flow-sensitive security type system [Hunt and Sands, POPL 2006]
- input/output check-points

Enforcement: Security Type System

Enforcement

- flow-sensitive security type system [Hunt and Sands, POPL 2006]
- input/output check-points

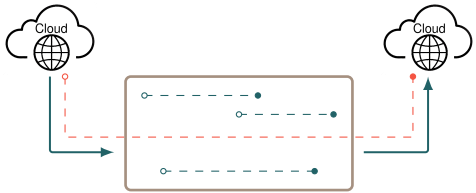


Information flows are allowed only if they do not affect the cloud

Enforcement: Security Type System

Enforcement

- flow-sensitive security type system [Hunt and Sands, POPL 2006]
- input/output check-points



Information flows are allowed only if they do not affect the cloud

Enforcement: Security Type System (contd.)

- Type system parametric in an initial (fixed) security typing Σ
- Judgments (for apps) of the form: $pc \vdash \Gamma\{\mathsf{Pld}\}\Gamma'$ — no flows from $\sigma \succ pc$ to pc
- Γ, Γ': typings before and after Pld — only services $\succ pc$ in Γ' may be changed by Pld

Enforcement: Security Type System (contd.)

- Type system parametric in an initial (fixed) security typing Σ
- Judgments (for apps) of the form: $pc \vdash \Gamma\{\mathsf{Pld}\}\Gamma'$ — no flows from $\sigma \succ pc$ to pc
- Γ, Γ': typings before and after Pld — only services $\succ pc$ in Γ' may be changed by Pld

$$(\text{Assign})\ \frac{\Gamma \vdash \mathsf{e} : \sigma}{pc \vdash \Gamma\{\mathsf{x} \leftarrow \mathsf{e}\}\Gamma[\mathsf{x} \mapsto \sigma \sqcup pc]}$$

$$(\text{Var})\ \frac{\Gamma(\mathsf{y}) = \sigma}{\Gamma \vdash \mathsf{y} : \sigma}$$

$$(\text{Read})\ \frac{\Sigma(\mathsf{y}) = \sigma}{\Gamma \vdash \mathsf{read}(\mathsf{y}) : \sigma}$$

$$(\text{Update})\ \frac{\Gamma(\mathsf{x}) \preccurlyeq \Sigma(\mathsf{x})}{pc \vdash \Gamma\{\mathsf{update}(\mathsf{x})\}\Gamma}$$

Enforcement: Security Type System (contd.)

- Type system parametric in an initial (fixed) security typing Σ
- Judgments (for apps) of the form: $pc \vdash \Gamma\{\mathsf{Pld}\}\Gamma'$ — no flows from $\sigma \succ pc$ to pc
- Γ, Γ': typings before and after Pld — only services $\succ pc$ in Γ' may be changed by Pld

$$(\text{Assign})\ \frac{\Gamma \vdash \mathsf{e} : \sigma}{pc \vdash \Gamma\{\mathsf{x} \leftarrow \mathsf{e}\}\Gamma[\mathsf{x} \mapsto \sigma \sqcup pc]}$$

look-up

$$(\text{Var})\ \frac{\Gamma(\mathsf{y}) = \sigma}{\Gamma \vdash \mathsf{y} : \sigma}$$ local

$$(\text{Read})\ \frac{\Sigma(\mathsf{y}) = \sigma}{\Gamma \vdash \mathsf{read}(\mathsf{y}) : \sigma}$$ cloud

$$(\text{Update})\ \frac{\Gamma(\mathsf{x}) \preccurlyeq \Sigma(\mathsf{x})}{pc \vdash \Gamma\{\mathsf{update}(\mathsf{x})\}\Gamma}$$

Enforcement: Security Type System (contd.)

- Type system parametric in an initial (fixed) security typing Σ
- Judgments (for apps) of the form: $pc \vdash \Gamma\{\mathsf{Pld}\}\Gamma'$ — no flows from $\sigma \succ pc$ to pc
- Γ, Γ': typings before and after Pld — only services $\succ pc$ in Γ' may be changed by Pld

$$(\textsc{Assign})\ \frac{\Gamma \vdash \mathsf{e} : \sigma}{pc \vdash \Gamma\{\mathsf{x} \leftarrow \mathsf{e}\}\Gamma[\mathsf{x} \mapsto \sigma \sqcup pc]}$$

look-up

$$(\textsc{Var})\ \frac{\Gamma(\mathsf{y}) = \sigma}{\Gamma \vdash \mathsf{y} : \sigma}$$ local

$$(\textsc{Read})\ \frac{\Sigma(\mathsf{y}) = \sigma}{\Gamma \vdash \mathsf{read}(\mathsf{y}) : \sigma}$$ cloud

$$(\textsc{Update})\ \frac{\Gamma(\mathsf{x}) \preccurlyeq \Sigma(\mathsf{x})}{pc \vdash \Gamma\{\mathsf{update}(\mathsf{x})\}\Gamma}$$

Soundness: well-typed systems are noninterfering

Conclusion

Contributions

Foundational framework for securing cross-app interactions

Contributions

Foundational framework for securing cross-app interactions

- Calculus for apps in IoT platforms

Contributions

Foundational framework for securing cross-app interactions

- Calculus for apps in IoT platforms
- Semantics condition for safe cross-app interactions
  - Enforcement: syntactic constraints for triggers and actions (Sound)
  - Extensions: implicit interactions and priorities between services

Contributions

Foundational framework for securing cross-app interactions

- Calculus for apps in IoT platforms
- Semantics condition for safe cross-app interactions
  - Enforcement: syntactic constraints for triggers and actions (Sound)
  - Extensions: implicit interactions and priorities between services
- Semantic condition for nonintereference in a system of apps
  - Enforcement: security flow-sensitive type system (Sound)

Contributions

Foundational framework for securing cross-app interactions

- Calculus for apps in IoT platforms
- Semantics condition for safe cross-app interactions
  - Enforcement: syntactic constraints for triggers and actions (Sound)
  - Extensions: implicit interactions and priorities between services
- Semantic condition for nonintereference in a system of apps
  - Enforcement: security flow-sensitive type system (Sound)

Thanks for attention!

Bonus slides

Safety extension: Implicit Interactions

Implicit interactions: semantically-related services with different syntax

Safety extension: Implicit Interactions

Implicit interactions: semantically-related services with different syntax

Therm

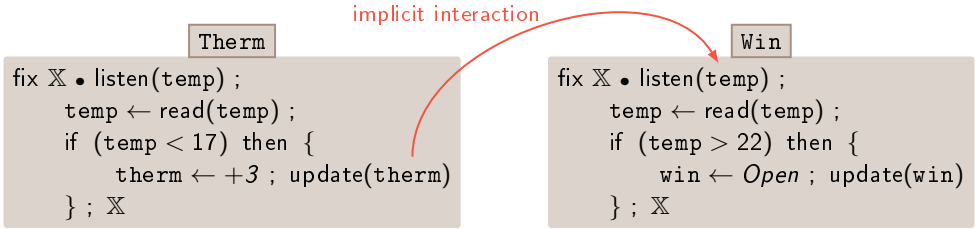
```
fix 𝕏 • listen(temp) ;
      temp ← read(temp) ;
      if (temp < 17) then {
          therm ← +3 ; update(therm)
      } ; 𝕏
```

Win

```
fix 𝕏 • listen(temp) ;
      temp ← read(temp) ;
      if (temp > 22) then {
          win ← Open ; update(win)
      } ; 𝕏
```

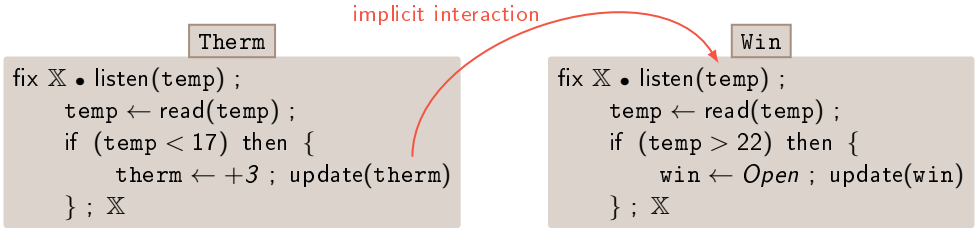
Safety extension: Implicit Interactions

Implicit interactions: semantically-related services with different syntax



Safety extension: Implicit Interactions

Implicit interactions: semantically-related services with different syntax



- Dependency policy $\Delta \subseteq \text{Services} \times \text{Services}$
- $y \in \mathfrak{clo}(\Delta, x)$, non-deterministic update on y
- Parametric LTS and hiding bisimulation

Safety extension: Priorities

The user intentionally allows some interactions

Safety extension: Priorities

The user intentionally allows some interactions

Priority policy:

- lattice of priority levels $\langle \mathrm{PL}, \sqsubseteq \rangle$
- priorities assignment Π to services

Safety extension: Priorities

The user intentionally allows some interactions

Priority policy:

- lattice of priority levels $\langle \mathrm{PL}, \sqsubseteq \rangle$
- priorities assignment Π to services

Noninteraction up-to priority level $\mathfrak{p} \in \mathrm{PL}$

- Indistinguishable behavior observing services at priority level greater than $\mathfrak{p}$
- We hide the updates of services lower than, or equal to, $\mathfrak{p}$
- Parametric LTS and hiding bisimulation