liquid information flow control

Nadia Polikarpova
WG2.8 2021

join work with Deian Stefan, Shachar Itzhaky, Jean Yang, Travis Hance, Armando Solar-Lezama
Facebook Security Breach Exposes Accounts of 50 Million Users

One of the challenges for Facebook's chief executive Mark Zuckerberg is convincing users that the company handles their data responsibly. Josh Edelson/Agence France-Presse — Getty Images

By Mike Isaac and Sheera Frenkel
Sept. 28, 2018

SAN FRANCISCO — Facebook, already facing scrutiny over how it handles the private information of its users, said on Friday that an attack on its computer network had exposed the personal information of nearly 50 million users.
This Is Almost Certainly James Comey’s Twitter Account

Digital security and its discontents—from Hillary Clinton’s emails to ransomware to Tor hacks—is in many ways one of the chief concerns of the contemporary FBI. So it makes sense that the bureau’s director, James Comey, would dip his toe into the digital torrent with a Twitter account. It also makes sense, given Comey’s high profile, that he would want that Twitter account to
Annual number of data breaches and exposed records in the United States from 2005 to 2019

(in millions)

Cyber crime: number of breaches and records exposed 2005-2019

Published by J. Clement, Mar 10, 2020

In 2019, the number of data breaches in the United States amounted to 1,473 with over 164.68 million sensitive records exposed.

Data breaches and exposed records - additional information

Data breaches have gained attention with the increasing use of digital files and companies and users large reliance on digital data. Even though data breaches happened before digitalization of information - for instance, looking at one's hard copy of medical files without authorization could be considered a breach - today they are more serious given the increasing use of digital data.
traditional policy enforcement

application code
traditional policy enforcement

application code
traditional policy enforcement

application code
ICF* frameworks

application code

policies

* Information Flow Control
IFC* frameworks

application code

policies

framework

* Information Flow Control
IFC* frameworks

application code

policies

framework

* Information Flow Control
existing IFC frameworks
existing IFC frameworks


application code

policies

framework

... static

dynamic

LIO [Stefan et al 2011] Jeeves [Yang et al 2012] ...
existing IFC frameworks

+ predictable behavior
+ no run-time overhead

- crashes / unpredictable behavior
- run-time overhead

static

dynamic

application code

policies

framework
existing IFC frameworks

+ predictable behavior
+ no run-time overhead
- rich policies require proof hints

application code
policies

framework

static
dynamic

- crashes / unpredictable behavior
  - run-time overhead
  + support for rich policies
our solution: lifty

+ predictable behavior
+ no run-time overhead
our solution: lifty

+ predictable behavior
+ no run-time overhead

application code

policies

static

Lifty
(Liquid Information Flow Types)
our solution: lifty

+ predictable behavior
+ no run-time overhead
+ automatic checking of rich policies

application code

policies

Lifty
(Liquid Information Flow Types)
our solution: lifty

+ predictable behavior
+ no run-time overhead
+ automatic checking of rich policies

application code

policies

(1) liquid types

Lifty
(Liquid Information Flow Types)
our solution: lifty

+ predictable behavior
+ no run-time overhead
+ automatic checking of rich policies

(1) liquid types

application code
policies

Lifty
(Liquid Information Flow Types)
our solution: lifty

+ predictable behavior
+ no run-time overhead
+ automatic checking of rich policies
+ leak repair

application code  

(1) liquid types

Lifty
(Liquid Information Flow Types)
our solution: lifty

+ predictable behavior
+ no run-time overhead
+ automatic checking of rich policies
+ leak repair

application code

policies

(1) liquid types
(2) type-driven program synthesis

Lifty
(Liquid Information Flow Types)
our solution: lifty

+ predictable behavior
+ no run-time overhead
+ automatic checking of rich policies
+ leak repair

application code

policies

(1) liquid types

(2) type-driven program synthesis

Lifty

(Liquid Information Flow Types)
1. IFC with liquid types
1. IFC with liquid types
2. programming in Lifty
1. IFC with liquid types
2. programming in Lifty
3. leak repair
1. IFC with liquid types
2. programming in Lifty
3. leak repair
liquid types*

\[
\text{Int}
\]

* [Rondon, Kawaguchi, Jhala. 2008]
liquid types*

\{ \nu: \text{Int} \mid 0 \leq \nu \}
liquid types*

\{ v : \text{Int} \mid 0 \leq v \}

natural numbers

* [Rondon, Kawaguchi, Jhala. 2008]
liquid types*

* [Rondon, Kawaguchi, Jhala. 2008]
security monads*

TIO Int

* [Russo, Claessen, Hughes, 2008], [Stefan et al 2011], ...
security monads*

sensitive computation that returns int

* [Russo, Claessen, Hughes, 2008], [Stefan et al 2011], ...
TIO = security monads + liquid types

TIO Int \langle u = alice \rangle \langle u = bob \rangle
TIO = security monads + liquid types

TIO Int \(\langle u = \text{alice}\rangle\) \(\langle u = \text{bob}\rangle\)

input label

output label
TIO = security monads + liquid types

TIO Int \langle u = alice \rangle \langle u = bob \rangle

may read data restricted to alice
TIO = security monads + liquid types

TIO Int <u = alice> <u = bob> may write to bob
TIO = security monads + liquid types

\[
\text{TIO Int } \langle u = \text{alice} \rangle \quad \langle u = \text{bob} \rangle
\]

the viewer

may write to bob
1. IFC with liquid types
   - the TIO monad
   - building TIO computations

2. programming in Lifty

3. leak repair
atomic TIO actions

model sources and sinks of sensitive data, e.g.:
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String \( \langle u \in [\text{alice, bob}] \rangle \langle \text{false} \rangle \)
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String $u \in [\text{alice, bob}]$ $\text{false}$
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String \langle u \in \{alice, bob\}\rangle \langle false\rangle
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String \(\forall u \in [\text{alice}, \text{bob}]\) \(\langle \text{false} \rangle\)

print :: \text{rec}:\text{User} \to \text{String} \to \text{TIO Unit} \(\langle \text{true} \rangle\) \(\langle u = \text{rec} \rangle\)
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String \(\langle u \in [alice, bob] \rangle \langle \text{false} \rangle\)

print :: rec:User \(\rightarrow\) String \(\rightarrow\) TIO Unit \(\langle \text{true} \rangle \langle u = \text{rec} \rangle\)
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String <u ∈ [alice, bob]> <false>

print :: rec:User → String → TIO Unit <true> <u = rec>
atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String \( \langle u \in [\text{alice, bob}] \rangle \langle \text{false} \rangle \)

print :: \text{rec}:User \rightarrow \text{String} \rightarrow \text{TIO Unit} \langle \text{true} \rangle \langle u = \text{rec} \rangle

dependent type
the TIO API
the TIO API

-- Return a pure value
return :: a → TIO a
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a →
    (a → TIO b) →
    TIO b
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a →
  (a → TIO b) →
  TIO b
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a <i> <o> →
    (a → TIO b <j> <i ∧ p>) →
    TIO b
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a <i> <o> →
    (a → TIO b <j> <i ∧ p>) →
    TIO b
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a <i> <o> →
     (a → TIO b <j> <i ∧ p>) →
     TIO b
the TIO API

-- Return a pure value
return :: a \to\ TIO a \langle\text{true}\rangle \langle\text{false}\rangle

-- Sequence two sensitive computations
bind :: TIO a \langle i \rangle \langle o \rangle \to

(a \to\ TIO b \langle j \rangle \langle i \land p \rangle) \to
TIO b
the TIO API

-- Return a pure value
return :: a → TIO a $\langle\text{true}\rangle \langle\text{false}\rangle$

-- Sequence two sensitive computations
bind :: TIO a $\langle i \rangle \langle o \rangle$ →
    (a → TIO b $\langle j \rangle \langle i \land p \rangle$) →
    TIO b $\langle i \land j \rangle$
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a <i> <o> →
  (a → TIO b <j> <i ∧ p>) →
  TIO b <i ∧ j> <o ∨ (i ∧ p)>
the TIO API

-- Return a pure value
return :: a → TIO a <true> <false>

-- Sequence two sensitive computations
bind :: TIO a <i> <o> →
    (a → TIO b <j> <i ∧ p>) →
    TIO b <i ∧ j> <o ∨ (i ∧ p)>

-- Safely downgrade a Boolean
downgrade :: ... -- in the paper
the TIO API

return :: a → TIO a <true> <false>

bind :: TIO a <i> <o> →
    (a → TIO b <j> <i ∧ p>) →
    TIO b <i ∧ j> <o ∨ (i ∧ p)>

downgrade :: ...

liftM :: ...    filterM :: ...
mapM :: ...    sortM :: ...
the TIO API

return :: a → TIO a <true> <false>

bind :: TIO a <i> <o> →
(a → TIO b <j> <i ∧ p>) →
TIO b <i ∧ j> <o ∨ (i ∧ p)>

downgrade :: ...

primitive operations (trusted)

liftM :: ...
filterM :: ...
mapM :: ...
sortM :: ...
the TIO API

return :: a → TIO a <true> <false>

bind :: TIO a <i> <o> →
(a → TIO b <j> <i ∧ p>) →
TIO b <i ∧ j> <o ∨ (i ∧ p)>

downgrade :: ...

primitive operations (trusted)

liftM :: ...
filterM :: ...
mapM :: ...
sortM :: ...

derived operations (verified)
1. IFC with liquid types
2. programming in Lifty
3. leak repair
running example: conference manager
## running example: conference manager

<table>
<thead>
<tr>
<th>#</th>
<th>title*</th>
<th>score</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive μ-types</td>
<td>1.8</td>
<td>accept</td>
</tr>
<tr>
<td>2</td>
<td>Naive Gradual Π-Calculus with Strong Normalization</td>
<td>0.8</td>
<td>reject</td>
</tr>
<tr>
<td>3</td>
<td>Abstract Impredicative Type Theory with Ownership Types</td>
<td>2.0</td>
<td>accept</td>
</tr>
<tr>
<td>4</td>
<td>Higher Type Theory with Primitive Recursion</td>
<td>1.5</td>
<td>accept</td>
</tr>
<tr>
<td>5</td>
<td>Liquid Information Flow Control</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>Standard Higher Calculus of Inductive Constructions with Full Type Inference</td>
<td>0.5</td>
<td>reject</td>
</tr>
</tbody>
</table>

*titles generated by http://type.systems/*
**running example: conference manager**

<table>
<thead>
<tr>
<th>#</th>
<th>title*</th>
<th>score</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive $\mu$-types</td>
<td>1.8</td>
<td>accept</td>
</tr>
<tr>
<td>2</td>
<td>Naive Gradual $\Pi$-Calculus with Strong Normalization</td>
<td>0.8</td>
<td>reject</td>
</tr>
<tr>
<td>3</td>
<td>Abstract Impredicative Type Theory with Coinductive Constructions with $\mu$-types</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>4</td>
<td>Higher Type</td>
<td>1.5</td>
<td>accept</td>
</tr>
<tr>
<td>5</td>
<td>Liquid Information Flow Control</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>Standard Higher Calculus of Inductive Constructions with Full Type Inference</td>
<td>0.5</td>
<td>reject</td>
</tr>
</tbody>
</table>

*titles generated by http://type.systems/*

**policy:** if conflicted, can’t see score until notification
**running example: conference manager**

<table>
<thead>
<tr>
<th>#</th>
<th>title*</th>
<th>score</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive $\mu$-types</td>
<td>1.8</td>
<td>accept</td>
</tr>
<tr>
<td>2</td>
<td>Naive Gradual $\Pi$-Calculus with Strong Normalization</td>
<td>0.8</td>
<td>reject</td>
</tr>
<tr>
<td>3</td>
<td>Abstract Impredicative Type Theory with Ownership Types</td>
<td>2.0</td>
<td>accept</td>
</tr>
<tr>
<td>4</td>
<td>Higher Type Theory with Primitive Recursion</td>
<td>1.5</td>
<td>accept</td>
</tr>
<tr>
<td>5</td>
<td>Liquid Information Flow Control</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>Standard Higher Calculus of Inductive Constructions with Full Type Inference</td>
<td>0.5</td>
<td>reject</td>
</tr>
</tbody>
</table>

*titles generated by http://type.systems/*
## running example: conference manager

<table>
<thead>
<tr>
<th>#</th>
<th>title*</th>
<th>score</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Abstract Impredicative Type Theory with Ownership Types</td>
<td>2.0</td>
<td>accept</td>
</tr>
<tr>
<td>5</td>
<td>Liquid Information Flow Control</td>
<td><em>c</em></td>
<td><em>c</em></td>
</tr>
<tr>
<td>1</td>
<td>Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive μ-types</td>
<td>1.8</td>
<td>accept</td>
</tr>
<tr>
<td>4</td>
<td>Higher Type Theory with Primitive Recursion</td>
<td>1.5</td>
<td>accept</td>
</tr>
<tr>
<td>2</td>
<td>Naive Gradual Π-Calculus with Strong Normalization</td>
<td>0.8</td>
<td>reject</td>
</tr>
<tr>
<td>6</td>
<td>Standard Higher Calculus of Inductive Constructions with Full Type Inference</td>
<td>0.5</td>
<td>reject</td>
</tr>
</tbody>
</table>

*titles generated by [http://type.systems/](http://type.systems/)*
# running example: conference manager

<table>
<thead>
<tr>
<th>#</th>
<th>title*</th>
<th>score</th>
<th>decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Abstract Impredicative Type Theory with Ownership Types</td>
<td>2.0</td>
<td>accept</td>
</tr>
<tr>
<td>5</td>
<td>Liquid Information Flow Control</td>
<td>c</td>
<td>accept</td>
</tr>
<tr>
<td>1</td>
<td>Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive ( \mu )-types</td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>4</td>
<td>Higher Type Theory</td>
<td>1.5</td>
<td>accept</td>
</tr>
<tr>
<td>2</td>
<td>Naive Gradual ( \Pi )-Calculus with Strong Normalization</td>
<td>0.8</td>
<td>reject</td>
</tr>
<tr>
<td>6</td>
<td>Standard Higher Calculus of Inductive Constructions with Full Type Inference</td>
<td>0.5</td>
<td>reject</td>
</tr>
</tbody>
</table>

*titles generated by http://type.systems/*
demo
1. IFC with liquid types
2. programming in Lifty
3. leak repair
do
    x ← getPaperScore ds pid
    print client (show x)
example

how does Lifty repair this leak?

do
  x ← getPaperScore ds pid
print client (show x)
example

how does Lifty repair this leak?

do
  x ← getPaperScore ds pid
  print client (show x)

key insight: use type errors to localize leaks!
example

bind (getPaperScore ds pid)
(\x . print client (show x))
step 1: liquid type checking

bind (getPaperScore ds pid)
  (\x . print client (show x))
step 1: liquid type checking

\[
\text{bind } \left( \text{getPaperScore } ds \text{ pid} \right)
\left( \lambda x . \text{print client } (\text{show } x) \right)
\]

\[
\text{TIO Int } \langle u \notin \text{conflicts } ds \text{ pid}\rangle \langle \text{false} \rangle
\]
step 1: liquid type checking

bind \( (\text{getPaperScore \ ds \ pid}) \)
\( (\backslash x . \text{print client (show x)}) \)

\[ \text{TIO Int} \ < u \not\in \text{conflicts ds pid} > < \text{false} > \]

\[ \text{TIO Unit} \ < \text{true} > < u = \text{client} > \]
step 1: liquid type checking

\[
\text{bind } (\text{getPaperScore ds pid})\\(\backslash x . \text{print client (show } x))\\
\]

\[
\text{TIO Int } <u \notin \text{conflicts ds pid}> <\text{false}>
\]

\[
\text{TIO Unit } <\text{true}> <u = \text{client}>
\]
step 1: liquid type checking

bind (getPaperScore ds pid) (\x . print client (show x))

TIO Int \( u \notin \text{conflicts ds pid} \) <false>

TIO Unit \( u = \text{client} \)
step 1: liquid type checking

bind (getPaperScore ds pid) (
x . print client (show x))

TIO Int <u ∉ conflicts ds pid> <false>

TIO Unit <true> <u = client>

SMT
step 1: liquid type checking

bind \( (\text{getPaperScore ds pid}) (\text{x} . \text{print client (show x)}) \)

\[ \text{TIO Int} \ <u \notin \text{conflicts ds pid}> \ <\text{false}> \]

\[ \text{TIO Unit} \ <\text{true}> \ <u = \text{client}> \]
step 1: liquid type checking

bind \( \text{getPaperScore ds pid} \) \\
(\( x \cdot \text{print client (show x)} \))

TIO Int \(< u \notin \text{conflicts ds pid} > < \text{false} >\)

SMT

TIO Unit \(< \text{true} > < u = \text{client} >\)
step 2: error localization

\[(\text{getPaperScore } ds \text{ pid})\]

\[\text{bind } (\text{getPaperScore } ds \text{ pid}) \equiv (\lambda x . \text{print client (show x)})\]

\[\text{TIO Int } <u \notin \text{conflicts ds pid}> <\text{false}>\]

\[\text{TIO Unit } <\text{true}> <u = \text{client}>\]
step 3: infer local specification

```
bind (getPaperScore ds pid)
(
x . print client (show x))
```

```
if ?? then getPaperScore ds pid else ??
```

TIO Unit <true> <u = client>
step 3: infer local specification

if ?? -- policy holds
then getPaperScore ds pid
else ??

bind \( \text{getPaperScore ds pid} \)
\( \langle x \ . \ \text{print client (show x)} \rangle \)

TIO Unit <true> <\( u = \text{client} \)>
step 3: infer local specification

```
if ?? -- policy holds
then getPaperScore ds pid
else ?? -- default

bind (getPaperScore ds pid)
(\x . print client (show x))

TIO Unit <true> <u = client>
```
step 3: infer local specification

```latex
\textbf{bind} \quad (\texttt{getPaperScore ds pid})

\quad (x \cdot \text{print client (show x)})

\textbf{TIO Unit} <true> <u = client>
```

```latex
\textbf{if} ?? \quad -- policy holds
\textbf{then} \texttt{getPaperScore ds pid}
\textbf{else} ?? \quad -- default
```

```
\textbf{TIO Int}
\quad <u = client>
\quad <false>
```
step 3: infer local specification

bind \( (\text{getPaperScore ds pid}) \)

\[(\backslash x . \text{print client (show x)}) \]

if ?? -- policy holds
  then getPaperScore ds pid
else ?? -- default

\[ \text{TIO Int} \ (\langle v = \text{client} \rangle \langle \text{false} \rangle) \]
step 4: type-driven synthesis*

{\text{bind} (\text{getPaperScore ds pid})}
(\backslash x . \text{print client} (\text{show} x))

\[
\text{if ?? then getPaperScore ds pid else ??}
\]

\text{TIO Int}
\langle \nu = \text{client} \rangle
\langle \text{false} \rangle

* [Polikarpova, Kuraj, Solar-Lezama. 2016]
step 4: type-driven synthesis*

```
do
    cs ← getPaperConflicts ds pid
    if not (elem client cs)
        then getPaperScore ds pid
    else ??
```

```
bind (getPaperScore ds pid)
    (\x . print client (show x))
```

* [Polikarpova, Kuraj, Solar-Lezama. 2016]
step 4: type-driven synthesis*

\[
\begin{align*}
\text{do} & \\
& \text{cs } \leftarrow \text{getPaperConflicts ds pid} \\
& \text{if not (elem client cs)} \\
& \quad \text{then getPaperScore ds pid} \\
& \text{else return 0}
\end{align*}
\]

\[
\text{bind (getPaperScore ds pid)}
\]

\[
(\forall x . \text{print client (show x)})
\]

* [Polikarpova, Kuraj, Solar-Lezama. 2016]
lifty

application code

policies

(1) liquid types

(2) type-driven program synthesis

40
lifty

(1) liquid types
(2) type-driven program synthesis

https://cseweb.ucsd.edu/~npolikarpova/lifty/
ongoing work

STORM

(Security-Typed ORM)
ongoing work

STORM
(Security-Typed ORM)
ongoing work

STORM
(Security-Typed ORM)

+ ORM
ongoing work

STORM
(Security-Typed ORM)

+ ORM + LiquidHaskell
ongoing work

STORM
(Security-Typed ORM)

+ ORM + LiquidHaskell

= secure web framework for Haskell