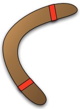
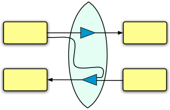
Updatable Security Views

Nate Foster
Benjamin Pierce
Steve Zdancewic

University of Pennsylvania

HCSS '09



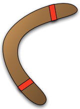


Updatable Security Views

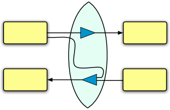
Nate Foster
Benjamin Pierce
Steve Zdancewic

University of Pennsylvania

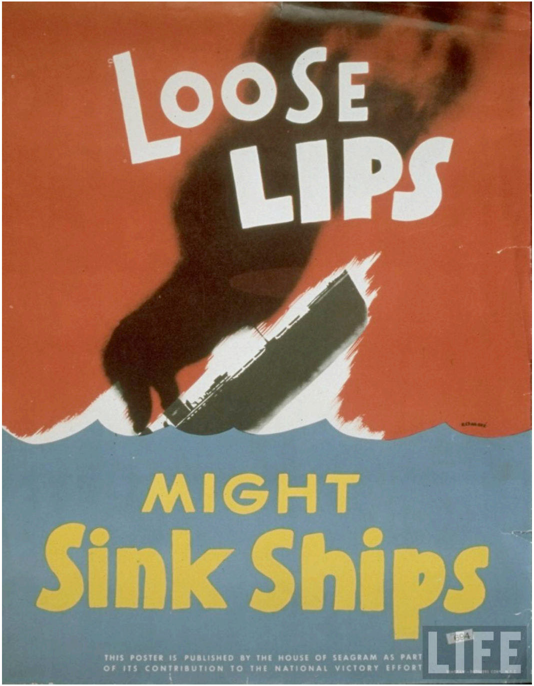
HCSS '09



PENN PL




TOP SECRET


LOOSE
LIPS
MIGHT
Sink Ships
THIS POSTER IS PUBLISHED BY THE HOUSE OF SEAGRAM AS PART
OF ITS CONTRIBUTION TO THE NATIONAL VICTORY EFFORT
LIFE

The Washington Post

"Pennsylvania yanks voter site after data leak"

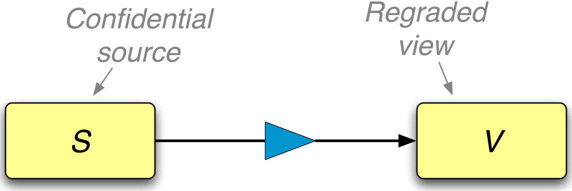
THE GLOBE AND MAIL
CANADA'S NATIONAL NEWSPAPER

"Passport applicant finds massive privacy breach"

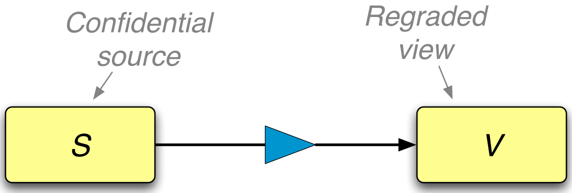
The New York Times

"Privacy issue complicates push to link medical data"

Security Views



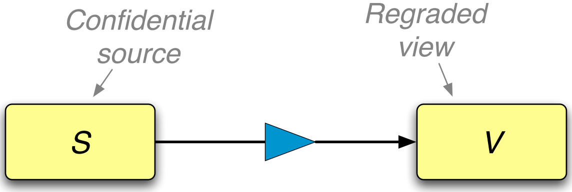
Security Views



✔ Robust: impossible to leak hidden data

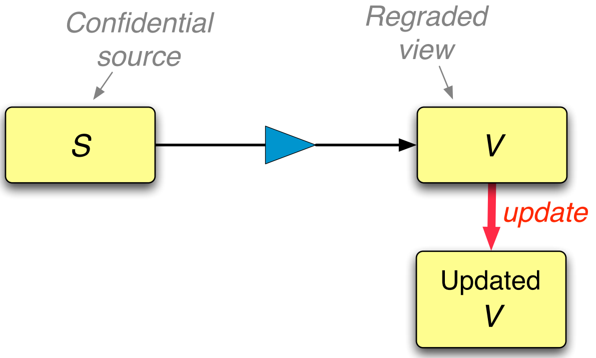
✔ Flexible: enforce fine-grained confidentiality policies

Security Views



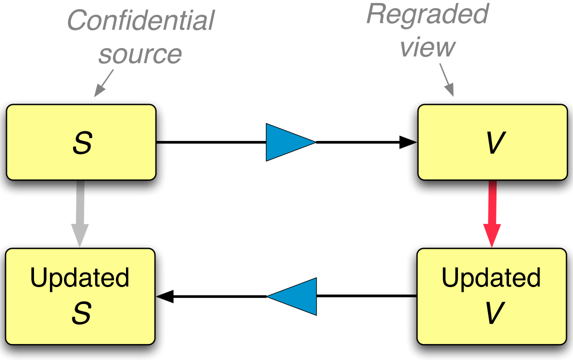
- ✔ Robust: impossible to leak hidden data
- ✔ Flexible: enforce fine-grained confidentiality policies
- ✗ Not usually updatable

Security Views



- ✔ Robust: impossible to leak hidden data
- ✔ Flexible: enforce fine-grained confidentiality policies
- ✗ Not usually updatable

Security Views



- ✔ Robust: impossible to leak hidden data
- ✔ Flexible: enforce fine-grained confidentiality policies
- ✗ Not usually updatable

Contributions

First steps toward a theory of updatable security views.

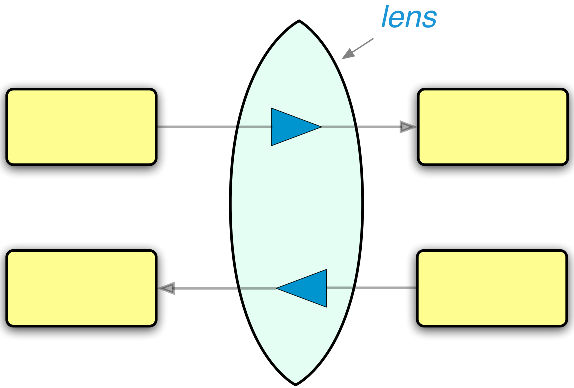
1. A generic semantic framework.
   - Building on previous work on lenses.
   - New non-interference laws provide additional guarantees about integrity.*
2. A concrete instantiation of these ideas in Boomerang, a language for writing bidirectional transformations over *ad hoc* string data.
   - Annotated regular expressions express integrity* policies.
   - Two enforcement mechanisms
     - A purely static program analysis
     - A hybrid static/dynamic analysis that can express a richer collection of integrity policies

* ...and confidentiality

Lenses

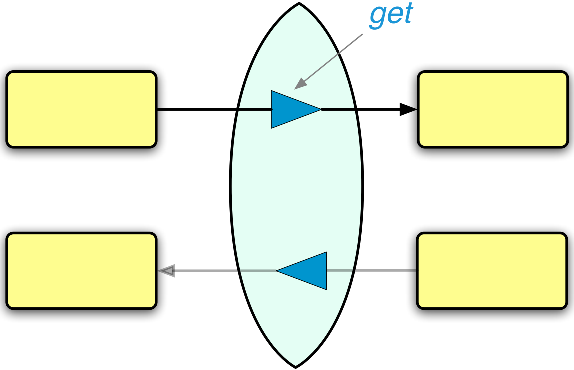
Bidirectional Transformations

In recent years, we've developed a number of programming languages for describing well-behaved bidirectional transformations called lenses.



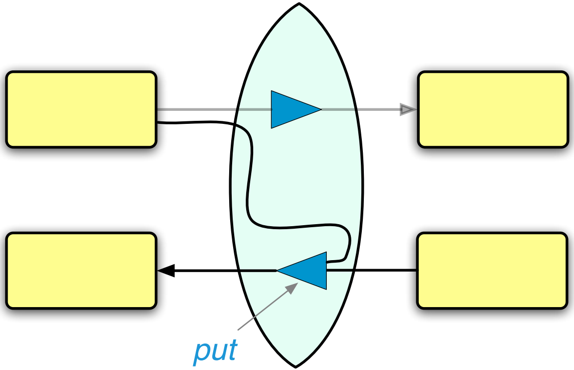
Lenses: Terminology

In recent years, we've developed a number of programming languages for describing well-behaved bidirectional transformations called lenses.



Lenses: Terminology

In recent years, we've developed a number of programming languages for describing well-behaved bidirectional transformations called lenses.



Lenses, Formally

A lens l mapping between a set S of sources and a set V of views is a pair of total functions

$$\begin{array}{lcl} l.\mathbf{get} & \in & S \rightarrow V \\ l.\mathbf{put} & \in & V \rightarrow S \rightarrow S \end{array}$$

Lenses, Formally

A lens l mapping between a set S of sources and a set V of views is a pair of total functions

$$\begin{aligned} l.\mathbf{get} &\in S \rightarrow V \\ l.\mathbf{put} &\in V \rightarrow S \rightarrow S \end{aligned}$$

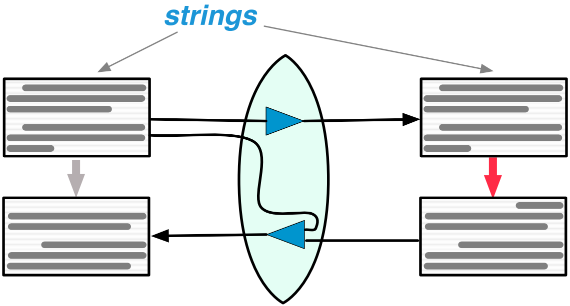
obeying "round-tripping" laws

$$l.\mathbf{get}\ (l.\mathbf{put}\ v\ s) = v \qquad (\textsc{PutGet})$$

$$l.\mathbf{put}\ (l.\mathbf{get}\ s)\ s = s \qquad (\textsc{GetPut})$$

for every $s \in S$ and $v \in V$.

Boomerang



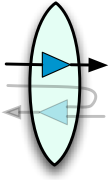
Data model: strings

Computation model: based on finite-state transducers

Types: regular expressions

Example: Redacting Calendars (Get)

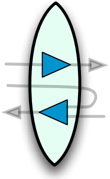
```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

Example: Redacting Calendars (Update)

```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



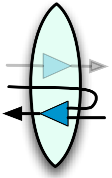
```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

```
08:30 BUSY
12:15 Lunch
15:00 BUSY
16:00 Meeting
```

Example: Redacting Calendars (Put)

```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```

```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

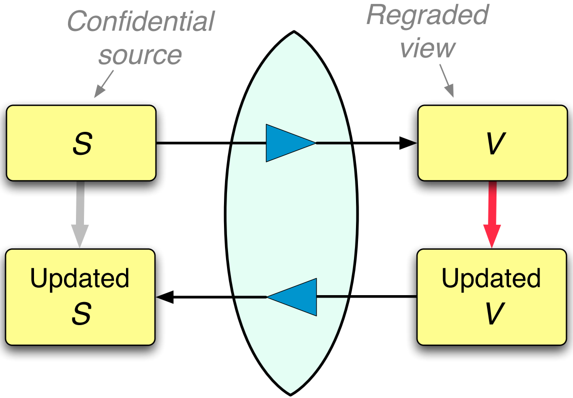


```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
 16:00 Meeting (Unknown)
```

```
08:30 BUSY
12:15 Lunch
15:00 BUSY
16:00 Meeting
```

Secure Lenses

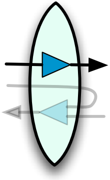
Requirements



1. Confidentiality: **get** does not leak secret data
2. Integrity: **put** does not taint trusted data

Example: Redacting Calendars (Get)

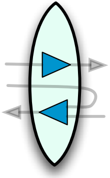
```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

Example: Redacting Calendars (Update II)

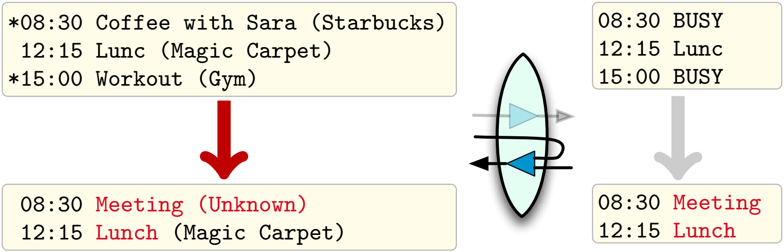
```
*08:30 Coffee with Sara (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

```
08:30 Meeting
12:15 Lunch
```

Example: Redacting Calendars (Put II)



Observe that propagating the update to the view back to the source forces **put** to modify some of the hidden source data:

- The entire appointment at 3pm.
- The description and location of the appointment at 8:30am.

Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

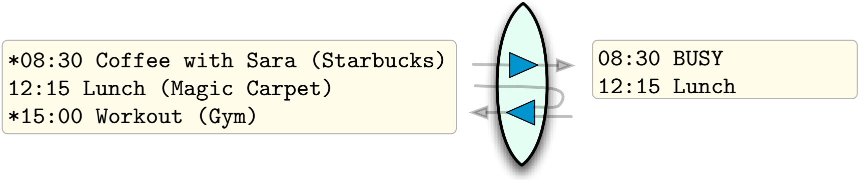
Integrity

Question: Should the (possibly untrusted) user of the view be allowed to modify hidden (possibly trusted) source data?

Answer: Maybe!

There are *many* alternatives, trading off which information in the source can be trusted against which information in the view can be edited.

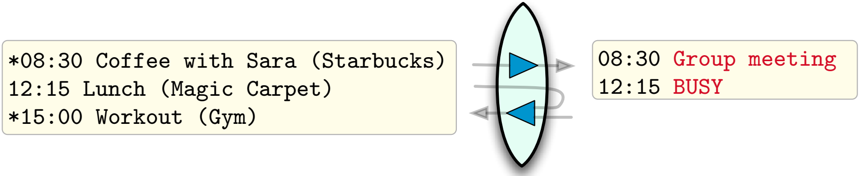
Some Integrity Policies



Policy: "Nothing is trusted" (whole source is tainted)

Effect: Arbitrary edits to the view are allowed; any hidden data in the source can be modified by **put**

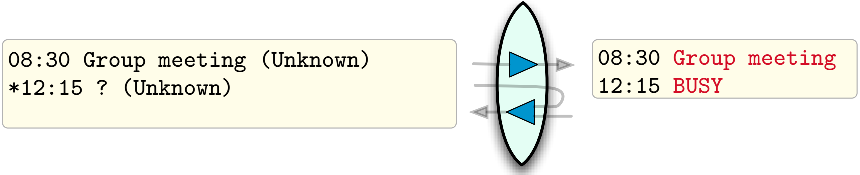
Some Integrity Policies



Policy: "Nothing is trusted" (whole source is tainted)

Effect: Arbitrary edits to the view are allowed; any hidden data in the source can be modified by **put**

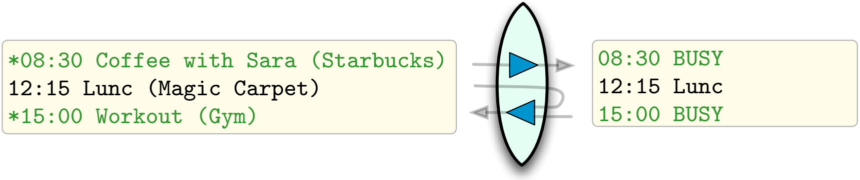
Some Integrity Policies



Policy: "Nothing is trusted" (whole source is tainted)

Effect: Arbitrary edits to the view are allowed; any hidden data in the source can be modified by **put**

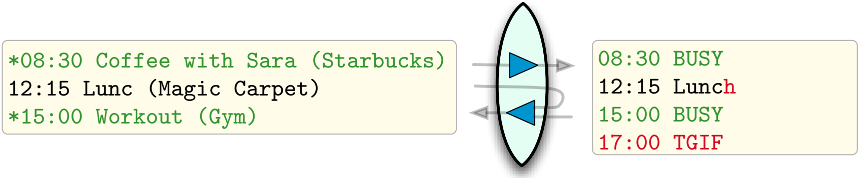
Some Integrity Policies



Policy: "Private appointments are trusted; public appointments are tainted"

Effect: OK to edit descriptions and add or delete public appointments, but not to add or delete private appointments or change between public and private

Some Integrity Policies

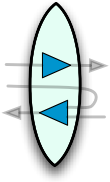


Policy: "Private appointments are trusted; public appointments are tainted"

Effect: OK to edit descriptions and add or delete public appointments, but not to add or delete private appointments or change between public and private

Some Integrity Policies

```
*08:30 Coffee with Sara (Starbucks)
12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
17:00 TGIF (Unknown)
```

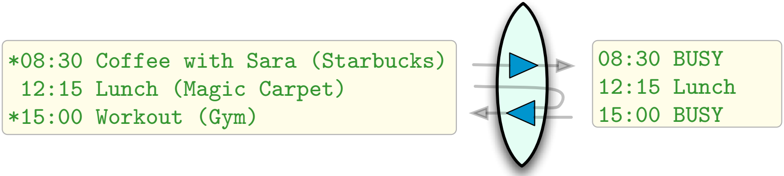


```
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
```

Policy: “Private appointments are trusted; public appointments are tainted”

Effect: OK to edit descriptions and add or delete public appointments, but not to add or delete private appointments or change between public and private

Another Integrity Policy

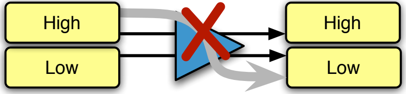


Policy: "Everything is trusted"

Effect: No edits are allowed

Non-interference

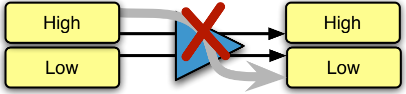
All these policies can be formulated in terms of non-interference.



A transformation is non-interfering if the “low” parts of the output do not depend on the “high” parts of the input.

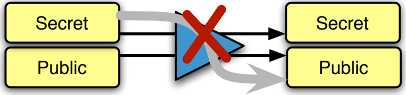
Non-interference — Confidentiality

All these policies can be formulated in terms of non-interference.



A transformation is non-interfering if the "low" parts of the output do not depend on the "high" parts of the input.

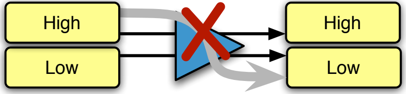
E.g., if the data contains both "secret" and "public" portions



then the secret parts of the input do not affect the public parts of the output.

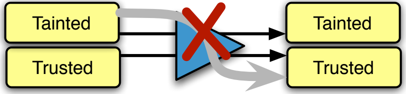
Non-interference — Integrity

All these policies can be formulated in terms of non-interference.



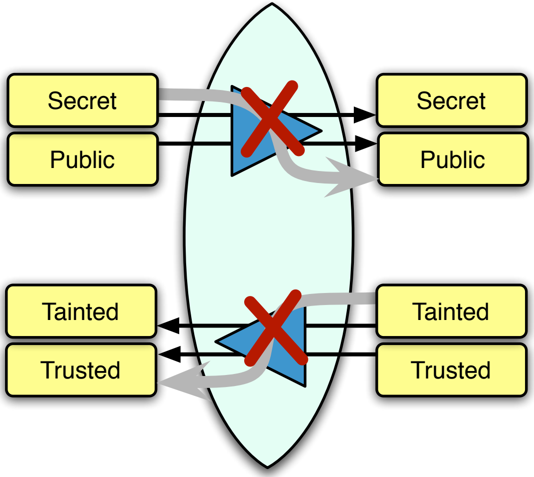
A transformation is non-interfering if the "low" parts of the output do not depend on the "high" parts of the input.

E.g., if the data contains "tainted" and "trusted" portions



then the tainted parts of the input do not affect the trusted parts of the output.

Secure Lenses



Static Enforcement

Labels

Fix a lattice of integrity labels, e.g.

Tainted

$\rotatebox{90}{$\leq$}$

Trusted

(again, eliding confidentiality...)

Annotated Regular Expressions

Mark up the source schema (a regular expression) to indicate which data is *Tainted* and which is *Trusted*.

$$\mathcal{R} ::= \emptyset \mid u \mid \mathcal{R}\cdot\mathcal{R} \mid \mathcal{R}|\mathcal{R} \mid \mathcal{R}* \mid \mathcal{R} : k$$

Annotated Regular Expressions

Mark up the source schema (a regular expression) to indicate which data is *Tainted* and which is *Trusted*.

$$\mathcal{R} ::= \emptyset \mid u \mid \mathcal{R}{\cdot}\mathcal{R} \mid \mathcal{R}|\mathcal{R} \mid \mathcal{R}* \mid \mathcal{R} : k$$

For example:

```
( (SPACE · TIME · DESC · LOCATION · NEWLINE):Tainted
| (ASTERISK · TIME · DESC · LOCATION · NEWLINE):Trusted)*
```

where

```
TIME = NUMBER{2} . COLON . NUMBER{2} . SPACE
DESC = [^\n()]* - (ANY . BUSY . ANY)
LOCATION = (SPACE . LPAREN . [^()]* . RPAREN)?
```

Equivalences

From the annotated source schema, read off an equivalence relation $\approx_k$, for each k in the lattice of integrity labels.

$s \approx_k s'$ is read "s and s' differ only on data that a user at level k has the authority to edit"

- $\approx_{Tainted}$ — "s and s' agree on trusted data"

```
*08:30 Coffee (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```

$\approx_{Tainted}$

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
```

- $\approx_{Trusted}$ — "s and s' agree on both trusted and tainted data" (i.e., they are identical)

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
```

$\approx_{Trusted}$

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
```

Example

```
*08:30 Coffee (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
 6:00 Dinner (Home)
```

Example

```
*08:30 Coffee (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```

↓ mark

```
*08:30 Coffee (Starbucks)
##########################
*15:00 Workout (Gym)
```

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
 6:00 Dinner (Home)
```

↓ mark

```
*08:30 Coffee (Starbucks)
###########################
*15:00 Workout (Gym)
###################
```

Example

```
*08:30 Coffee (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```

mark

```
*08:30 Coffee (Starbucks)
##########################
*15:00 Workout (Gym)
```

erase

```
*08:30 Coffee (Starbucks)
*15:00 Workout (Gym)
```

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
 6:00 Dinner (Home)
```

mark

```
*08:30 Coffee (Starbucks)
###########################
*15:00 Workout (Gym)
###################
```

erase

```
*08:30 Coffee (Starbucks)
*15:00 Workout (Gym)
```

Example

```
*08:30 Coffee (Starbucks)
 12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
```

↓ mark

```
*08:30 Coffee (Starbucks)
##########################
*15:00 Workout (Gym)
```

↓ erase

```
*08:30 Coffee (Starbucks)
*15:00 Workout (Gym)
```

=

```
*08:30 Coffee (Starbucks)
 12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
 6:00 Dinner (Home)
```

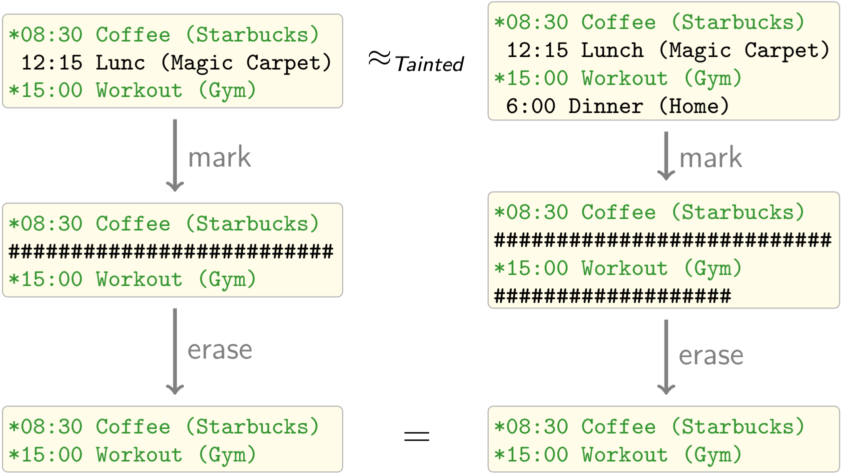
↓ mark

```
*08:30 Coffee (Starbucks)
###########################
*15:00 Workout (Gym)
###################
```

↓ erase

```
*08:30 Coffee (Starbucks)
*15:00 Workout (Gym)
```

Example


*08:30 Coffee (Starbucks)
12:15 Lunc (Magic Carpet)
*15:00 Workout (Gym)
≈Tainted
*08:30 Coffee (Starbucks)
12:15 Lunch (Magic Carpet)
*15:00 Workout (Gym)
6:00 Dinner (Home)
mark
mark
*08:30 Coffee (Starbucks)
##########################
*15:00 Workout (Gym)
*08:30 Coffee (Starbucks)
###########################
*15:00 Workout (Gym)
###################
erase
erase
*08:30 Coffee (Starbucks)
*15:00 Workout (Gym)
=
*08:30 Coffee (Starbucks)
*15:00 Workout (Gym)

Static Analysis

Use a simple static analysis to push the annotated source schema through the lens program to obtain an annotated view schema.

Static Analysis

```
(  (SPACE . TIME . DESC . LOCATION . NEWLINE) : Tainted
 | (ASTERISK . TIME . DESC . LOCATION . NEWLINE) : Trusted )*
```

```
let public : lens =
  del ( SPACE ) .
  copy ( TIME . DESC ) .
  del ( LOCATION ) .
  copy ( NEWLINE )

let private : lens =
  del ASTERISK  .
  copy ( TIME ) .
  ( ( DESC . LOCATION ) <-> "BUSY" ) .
  copy NEWLINE

let redact : lens =
  public* . ( private . public* )*
```

↓

```
(  (TIME . DESC . NEWLINE) : Tainted
 | (TIME . BUSY . NEWLINE) : Trusted )*
```

Static Analysis

Use a simple static analysis to push the annotated source schema through the lens program to obtain an annotated view schema.

From the annotated view schema, read off an equivalence on views.

Example

```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

```
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
```

Example

```
08:30 BUSY
12:15 Lunc
15:00 BUSY
```

↓ mark

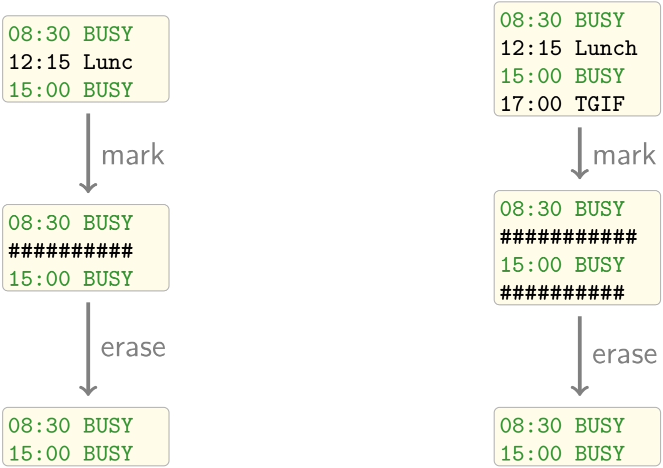
```
08:30 BUSY
##########
15:00 BUSY
```

```
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
```

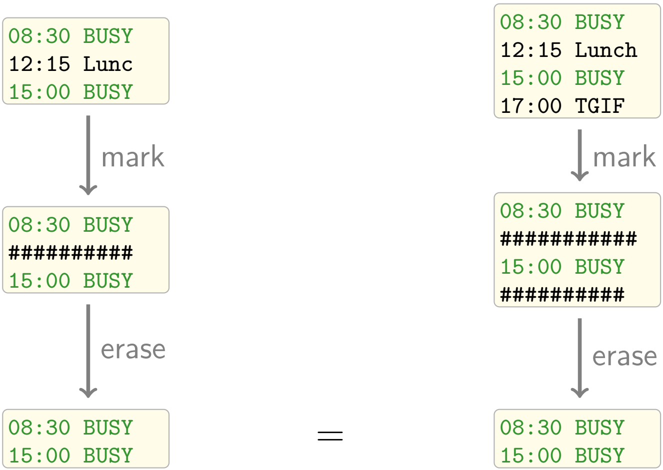
↓ mark

```
08:30 BUSY
###########
15:00 BUSY
##########
```

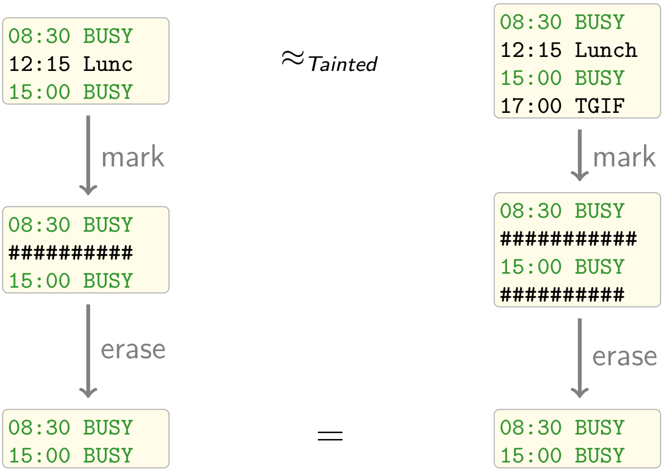
Example


08:30 BUSY
12:15 Lunc
15:00 BUSY
mark
08:30 BUSY
##########
15:00 BUSY
erase
08:30 BUSY
15:00 BUSY
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
mark
08:30 BUSY
###########
15:00 BUSY
##########
erase
08:30 BUSY
15:00 BUSY

Example


08:30 BUSY
12:15 Lunc
15:00 BUSY
mark
08:30 BUSY
##########
15:00 BUSY
erase
08:30 BUSY
15:00 BUSY
=
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
mark
08:30 BUSY
###########
15:00 BUSY
##########
erase
08:30 BUSY
15:00 BUSY

Example



08:30 BUSY
12:15 Lunc
15:00 BUSY
≈ Tainted
08:30 BUSY
12:15 Lunch
15:00 BUSY
17:00 TGIF
mark
mark
08:30 BUSY
##########
15:00 BUSY
08:30 BUSY
###########
15:00 BUSY
##########
erase
erase
08:30 BUSY
15:00 BUSY
=
08:30 BUSY
15:00 BUSY

Secure Lenses, Formally

The expectation that "*Tainted* inputs to **put** should not affect *Trusted* outputs" can now be expressed by generalizing the GetPut law...

$$l.\mathbf{put}\ (l.\mathbf{get}\ s)\ s = s \qquad \text{(GetPut)}$$

... like this:

$$\frac{v \approx_k (l.\mathbf{get}\ s)}{l.\mathbf{put}\ v\ s \approx_k s} \qquad \text{(GetPutSecure)}$$

We prove in the paper that our static analysis guarantees this new law.

(We also keep the original PutGet law and add a similar law for confidentiality.)

Usage Scenario

We can now maintain integrity of the source data after updates as follows:

1. Start with source s
2. Alice (the Owner of the source) uses **get** to create a view v
3. Alice gives v to Eve (an untrusted user)
4. Eve edits v to produce v' and gives v' back to Alice
5. Alice checks that v and v' agree on trusted data (i.e., $v \approx_{Tainted} v'$)
   - If so, Alice replaces s with **put** s v'
   - If not, Alice refuses the update
6. Safety theorem for the static analysis guarantees $s \approx_{Tainted} s'$ — i.e., s and s' agree on trusted data.

:-)

The PutPut Law, Redux

The following law can be derived:

$$\frac{v' \approx_k v \approx_k (l.\mathbf{get}\ s)}{l.\mathbf{put}\ v'\ (l.\mathbf{put}\ v\ s) \approx_k l.\mathbf{put}\ v'\ s}$$

This law says that the **put** function must have no "side-effects" on trusted source data.

It generalizes the "constant complement" condition, the gold standard for correct view update in databases.

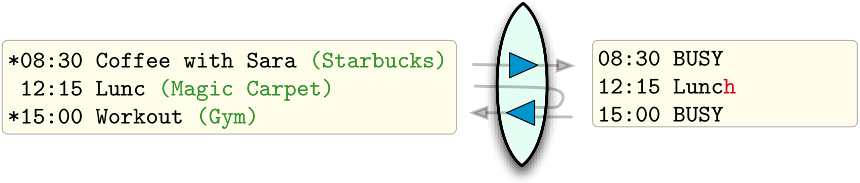
Dynamic Enforcement

Static Analyses Are Conservative

There are useful integrity policies for which the static enforcement method is too conservative, disallowing too many edits.

For example...

Another Possible Integrity Policy



Policy: "Event locations are trusted"

Desired effect: Edits to the view are allowed as long as they do not change its length or change public events to private or vice versa.

Problem: No way to achieve this effect by marking regions of the view as `Trusted` / `Tainted`.

Dynamic Approach

1. Extend secure lenses with dynamic tests that check if the **put** function can safely handle a given source and view:

$$l.\textbf{safe} \in \mathcal{Q} \to V \to S \to \mathbb{B}$$

2. Replace GetPutSecure with the following law:

$$\frac{l.\textbf{safe}\ q\ v\ s}{l.\textbf{put}\ v\ s \approx_q s} \qquad (\textsc{GetPutDyn})$$

3. Change usage scenario...

(see the paper for the confidentiality side of the story)

Usage Scenario (dynamic version)

We can now maintain integrity of the source data after updates as follows:

1. Start with source s
2. Alice (the Owner of the source) uses **get** to create a view v
3. Alice gives v to Eve (an untrusted user)
4. Eve edits v to produce v' and gives v' back to Alice
5. Alice checks l.**safe** *Tainted* v' s
   - If *true*, Alice replaces s with **put** s v'
   - If *false*, Alice refuses the update
6. Safety theorem for the static analysis guarantees $s \approx_{Tainted} s'$ — i.e., s and s' agree on trusted data.

:-)

Finishing up...

Examples Under Construction

- A multi-level wiki, inspired by Intellipedia and by the Galois *Tearline Wiki* project
- Tool for sharing / synchronizing calendars, bibliographic databases, etc. with partial visibility
- ... Suggestions welcome!

Other Ongoing Work

- Implementation (of type system)
- Security implications of rich alignment strategies
- Richer lattices (decentralized label model)
- Provenance / auditing

Thank You!

Want to play? Boomerang is available for download.

- Source code (LGPL)
- Precompiled binaries
- Research papers
- Tutorial and demos

http://www.seas.upenn.edu/~harmony/