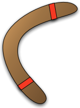
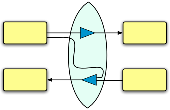
Updatable Security Views

Nate Foster
Benjamin Pierce
Steve Zdancewic

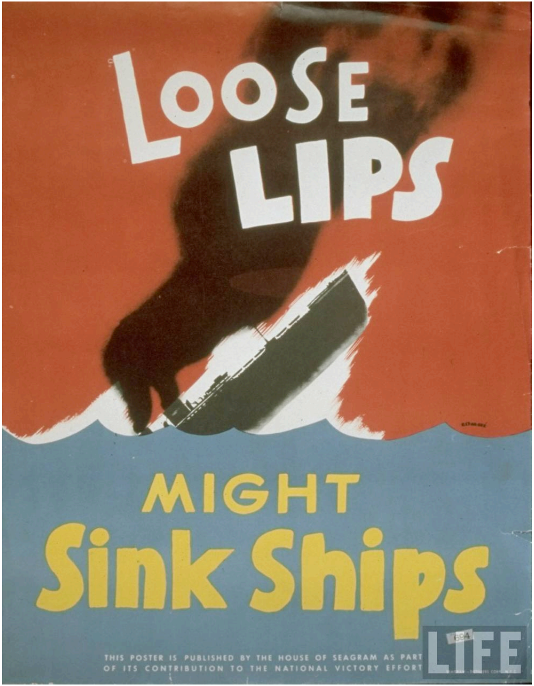
University of Pennsylvania

IBM PLDay '09






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"

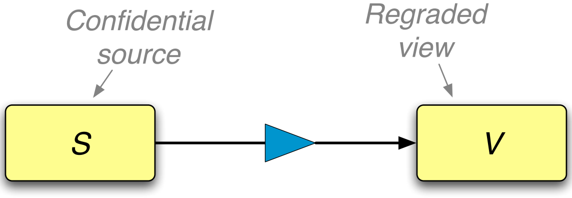
THE GLOBE AND MAIL
CANADA'S NATIONAL NEWSPAPER

"Passport applicant finds massive privacy breach"

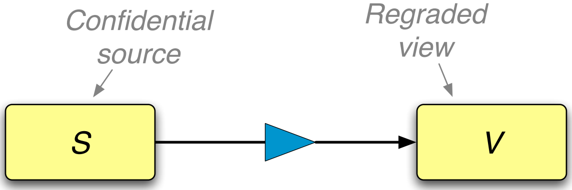
The New York Times

"Privacy issue complicates push to link medical data"

Security Views



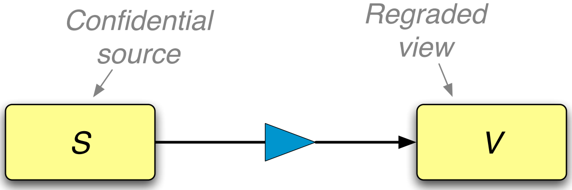
Security Views



✔ Robust: impossible to leak hidden data

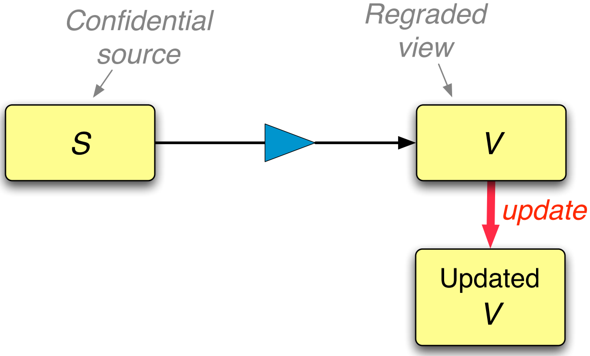
✔ Flexible: enforce fine-grained confidentiality policies

Security Views



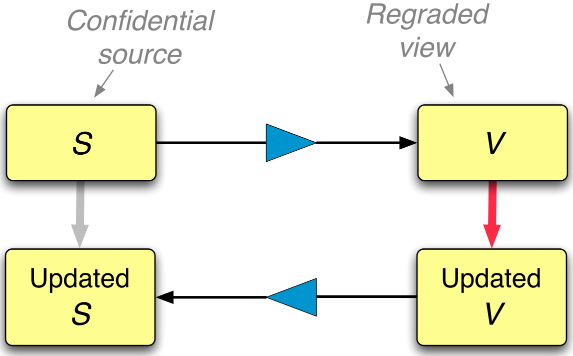
- ✔ Robust: impossible to leak hidden data
- ✔ Flexible: enforce fine-grained confidentiality policies
- ✗ Not usually updatable
- ✗ No separate specification of confidentiality policy

Updatable Security Views



- ✔ Robust: impossible to leak hidden data
- ✔ Flexible: enforce fine-grained confidentiality policies
- ✗ Not usually updatable
- ✗ No separate specification of confidentiality policy

Updatable Security Views



- ✔ Robust: impossible to leak hidden data
- ✔ Flexible: enforce fine-grained confidentiality policies
- ✗ Not usually updatable
- ✗ No separate specification of confidentiality policy

This Talk

A generic framework for building updatable security views.

- Extends previous work on lenses.
- New non-interference laws provide additional guarantees about confidentiality and integrity.

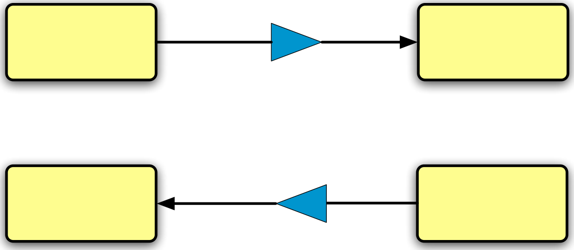
A concrete instantiation of these ideas in Boomerang, a language for writing lenses on strings.

- Annotated regular expressions express confidentiality and integrity policies.

Lenses

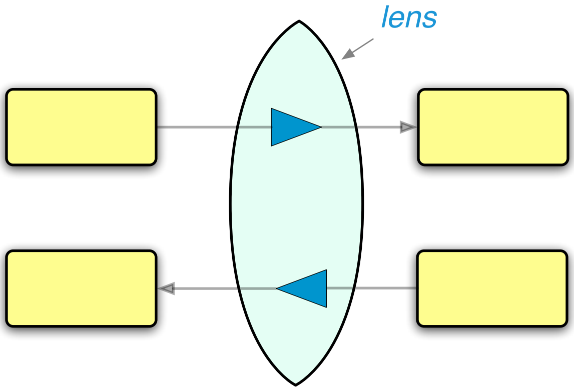
Bidirectional Transformations

For a view to be updatable, the program that defines it needs to be bidirectional.



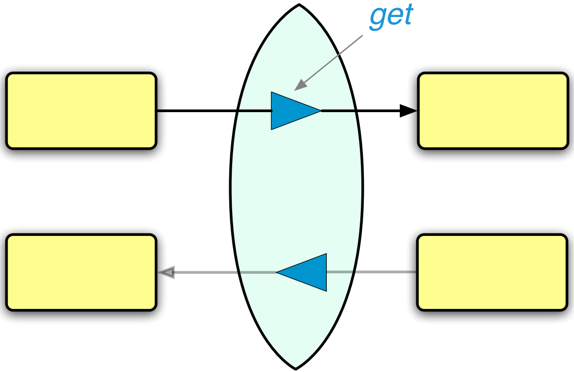
Lenses

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



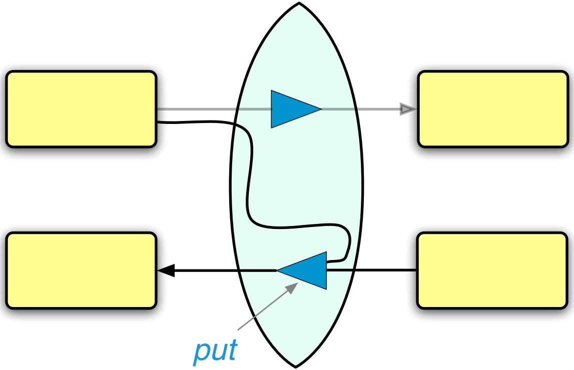
Lenses: Terminology

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



Lenses: Terminology

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



Semantics

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

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

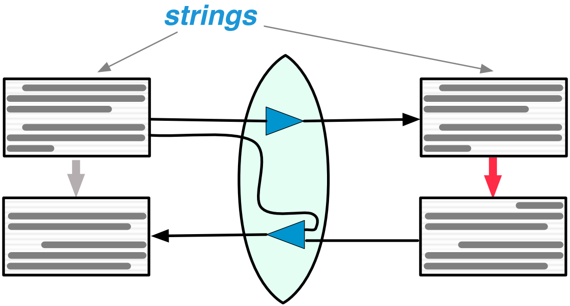
obeying "round-tripping" laws

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

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

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

Boomerang



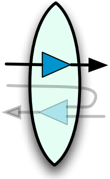
Data model: strings

Computation model: based on finite-state transducers

Types: regular expressions

Example: Redacting Calendars (Get)

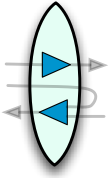
```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



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

Example: Redacting Calendars (Update)

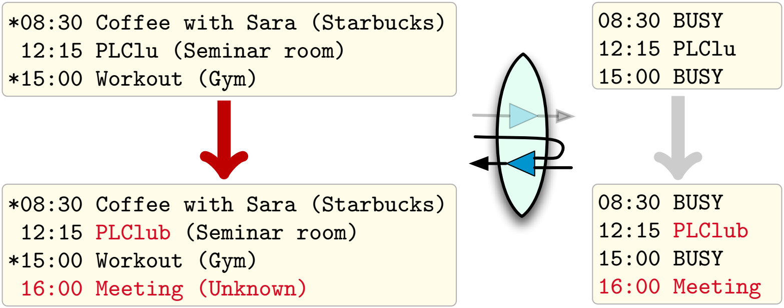
```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



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

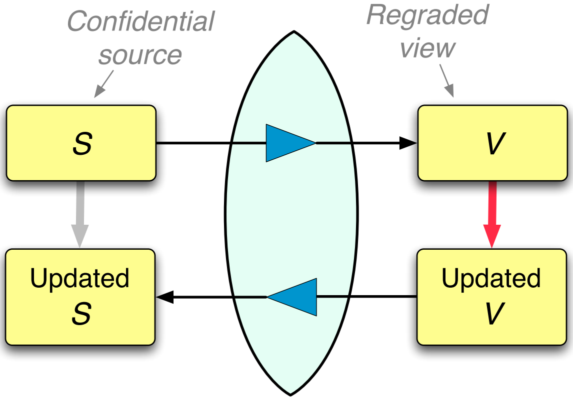
```
08:30 BUSY
12:15 PLClub
15:00 BUSY
16:00 Meeting
```

Example: Redacting Calendars (Put)



Secure Lenses

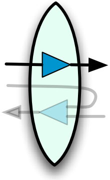
Requirements



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

Example: Redacting Calendars (Get)

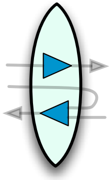
```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



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

Example: Redacting Calendars (Update II)

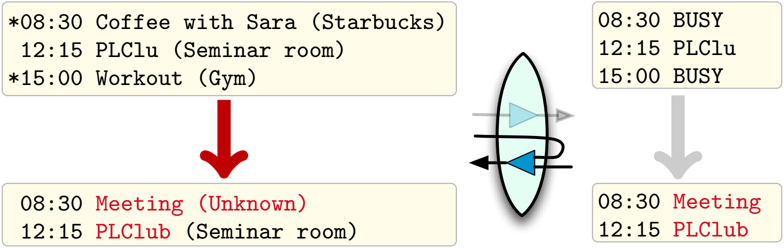
```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



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

```
08:30 Meeting
12:15 PLClub
```

Example: Redacting Calendars (Put II)



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

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

Integrity

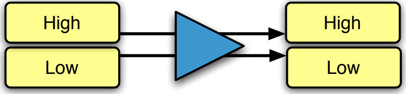
Question: should the (potentially untrusted) user of the view be allowed to modify hidden (potentially confidential) source data?

Answer: It depends $\rightarrow$ we need to be able to formulate and choose between integrity policies like

- "These appointments in the source may be altered"
- "These appointments in the source may not be altered (and so the view must not be modified in certain ways)"

Non-interference

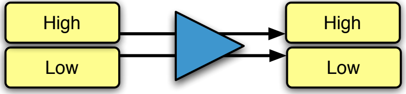
Both requirements can both be formulated as non-interference.



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

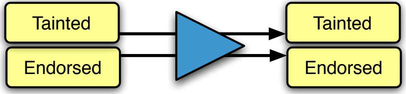
Non-interference

Both requirements can both be formulated as non-interference.



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

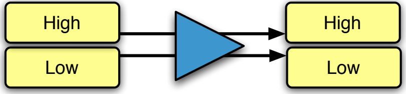
E.g., if the data contains "tainted" and "endorsed" portions



then non-interference says that the tainted parts of the input do not affect the endorsed parts of the output.

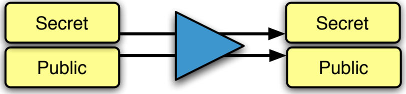
Non-interference

Both requirements can both be formulated as non-interference.



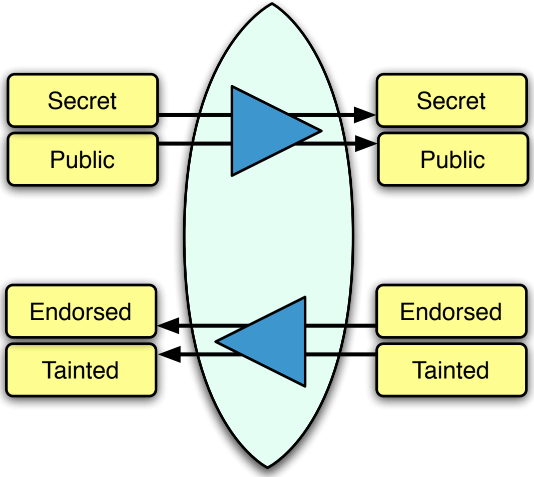
A transformation is non-interfering if the low-security parts of the output do not depend on the high-security parts of the input.

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

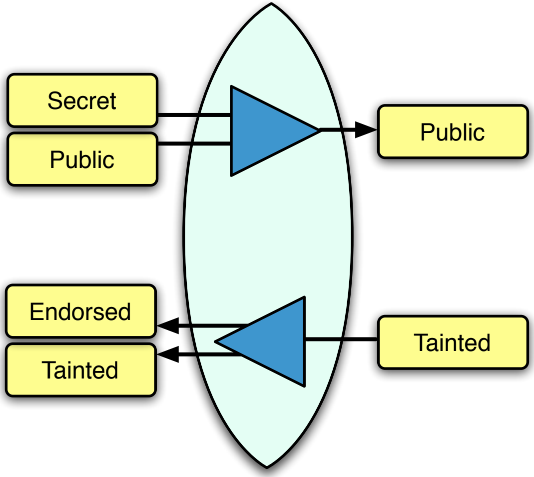


then non-interference says that the secret parts of the input do not affect the public parts of the output.

Secure Lenses



Secure Lenses



Semantics of Secure Lenses

Fix a family of equivalence relations on S and V

- $\sim_k$ — "agree on k-public data"
- $\approx_k$ — "agree on k-endorsed data"

that capture notions of high and low-security data.

Semantics of Secure Lenses

Fix a family of equivalence relations on S and V

- $\sim_k$ — "agree on k-public data"
- $\approx_k$ — "agree on k-endorsed data"

that capture notions of high and low-security data.

A secure lens obeys refined behavioral laws:

$$\frac{s \sim_k s'}{l.\mathbf{get}\ s \sim_k l.\mathbf{get}\ s'} \qquad (\textsc{GetNoLeak})$$

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

(as well as the original PutGet law).

Protocol for Using a Secure Lens

Before the owner of the source allows the user of the view to propagate an update using **put**, they check that the old and new views agree on endorsed data.

The GetPut law

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

ensures that endorsed data in the source is preserved.

Enforces high-level integrity policies such as

- "These appointments in the source may be altered"
- "These appointments in the source may not be altered..."

For Experts: the PutPut Law

The following law can be derived.

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

It says that the **put** function must have no "side-effects" on endorsed source data.

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

Syntax for Secure Lenses

In Boomerang, we describe the $\sim_k$ and $\approx_k$ equivalence relations using annotated regular expressions.

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

The relations are based on an intuitive notion of "erasing" characters inaccessible to a k-observer...

Syntax for Secure Lenses

In Boomerang, we describe the $\sim_k$ and $\approx_k$ equivalence relations using annotated regular expressions.

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

The relations are based on an intuitive notion of "erasing" characters inaccessible to a *k*-observer...

See paper for:

- A secure lens version of Boomerang's type system that tracks information flow—in two directions!
- An extension to this type system that uses a combination of static and dynamic checks to ensure integrity.

Conclusion

Summary:

- Data processing is a fertile area for exploring language-based approaches to security.
- Secure lenses provide a reliable framework for constructing updatable security views.
- Mechanisms for ensuring the integrity of data are critical.

Ongoing Work:

- Type system implementation
- Applications
- Other semantics for annotated regular types
- Investigate expressiveness vs. precision

Thank You!

Collaborators: Benjamin Pierce and Steve Zdancewic.

Want to play? Boomerang is available for download.

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

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

Dynamic Approach

In the paper we show how to extend secure lenses with dynamic tests that check if the **put** function can safelty handle a given source and view:

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

We replace GetPut with the following law:

$$\frac{l.\textbf{safe}\ (p, q)\ v\ s}{l.\textbf{put}\ v\ s \approx_q s} \quad (\textsc{GetPut})$$

We add a non-interference law stipulating that the **safe** function must not leak secrets:

$$\frac{v \sim_p v' \qquad s \sim_p s'}{l.\textbf{safe}\ (p, q)\ v\ s = l.\textbf{safe}\ (p, q)\ v'\ s'} \ (\textsc{SafeNoLeak})$$