Updatable Security Views

Nate Foster
Benjamin Pierce
Steve Zdancewic

University of Pennsylvania

IBM PLDay ’09
LOOSE LIPS
MIGHT Sink Ships

THIS POSTER IS PUBLISHED BY THE HOUSE OF SEAGRAM AS PART OF ITS CONTRIBUTION TO THE NATIONAL VICTORY EFFORT
“Pennsylvania yanks voter site after data leak”

“Passport applicant finds massive privacy breach”

“Privacy issue complicates push to link medical data”
Security Views

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

Confidential source

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

Regraded view

update

S

V

Updated V
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.
In recent years, we have developed a number of bidirectional programming languages for describing certain well-behaved transformations called lenses.
In recent years, we have developed a number of bidirectional programming languages for describing certain well-behaved transformations called lenses.
A lens \( l \) mapping between a set \( S \) of sources and \( V \) of view is a pair of total functions

\[
\begin{align*}
l.\text{get} & \in S \rightarrow V \\
l.\text{put} & \in V \rightarrow S \rightarrow S
\end{align*}
\]

obeying “round-tripping” laws

\[
\begin{align*}
l.\text{get} (l.\text{put} \; \nu \; s) & = \nu & \text{(PutGet)} \\
l.\text{put} (l.\text{get} \; s) \; s & = s & \text{(GetPut)}
\end{align*}
\]

for every \( s \in S \) and \( \nu \in V \).
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)

<table>
<thead>
<tr>
<th>Time</th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:30</td>
<td>Coffee with Sara (Starbucks)</td>
</tr>
<tr>
<td>12:15</td>
<td>PLClub (Seminar room)</td>
</tr>
<tr>
<td>*15:00</td>
<td>Workout (Gym)</td>
</tr>
<tr>
<td>08:30</td>
<td>BUSY</td>
</tr>
<tr>
<td>12:15</td>
<td>PLClub</td>
</tr>
<tr>
<td>15:00</td>
<td>BUSY</td>
</tr>
</tbody>
</table>

*Note: The original times and events have been redacted to indicate an update.*
Example: Redacting Calendars (Put)

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

08:30 BUSY
12:15 PLClub
15:00 BUSY

*08:30 Coffee with Sara (Starbucks)
  12:15 PLClub (Seminar room)
*15:00 Workout (Gym)
  16:00 Meeting (Unknown)

08:30 BUSY
12:15 PLClub
15:00 BUSY
16:00 Meeting
Secure Lenses
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 PLCtu (Seminar room)
*15:00 Workout (Gym)

08:30 BUSY
12:15 PLCtu
15:00 BUSY

08:30 Meeting
12:15 PLCtu
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.
Question: should the (potentially untrusted) user of the view be allowed to modify hidden (potentially confidential) source data?

Answer: It depends → 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.
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:

A secure lens obeys refined behavioral laws:

\[
\frac{s \sim_k s'}{l.\text{get } s \sim_k l.\text{get } s'}
\]

(GET\text{NO\text{LEAK})}

\[
\frac{v \approx_k (l.\text{get } s)}{l.\text{put } v \ s \approx_k s}
\]

(GET\text{PUT})

(as well as the original PUT\text{GET} law).
Protocol for Using a Secure Lens

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

The \texttt{GetPut} law

\[
\frac{v \approx_k (l.\texttt{get} s)}{l.\texttt{put} v s \approx_k s}
\]

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.

\[ v' \approx_k v \approx_k (l.\text{get } s) \]

\[ l.\text{put } v' \ (l.\text{put } v \ s) \approx_k l.\text{put } v' \ s \]

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.

$$ R ::= \emptyset \mid u \mid R \cdot R \mid R \mid R \mid R^* \mid R : k $$

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.

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

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 \textbf{put} function can safely handle a given source and view:

\[
\text{\textit{l.safe} } \in (P \times Q) \rightarrow V \rightarrow S \rightarrow B
\]

We replace \texttt{GetPut} with the following law:

\[
\frac{\text{\textit{l.safe} } (p, q) \ v \ s}{\text{\textit{l.put} } v \ s \ \approx_q s} \quad \text{(GetPut)}
\]

We add a non-interference law stipulating that the \textbf{safe} function must not leak secrets:

\[
\frac{v \ \sim_p v' \quad s \ \sim_p s'}{\text{\textit{l.safe} } (p, q) \ v \ s = \text{\textit{l.safe} } (p, q) \ v' \ s'} \quad \text{(SafeNoLeak)}
\]