Object Capabilities and Isolation of Untrusted Web Applications

Ankur Taly

Dept. of Computer Science, Stanford University

Joint work with Sergio Maffeis (Imperial College London) and John C. Mitchell (Stanford University)

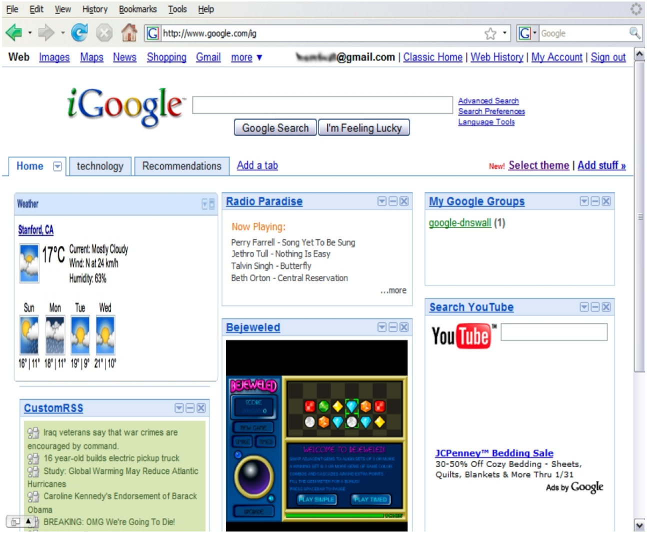
Outline

What are Mashups ?

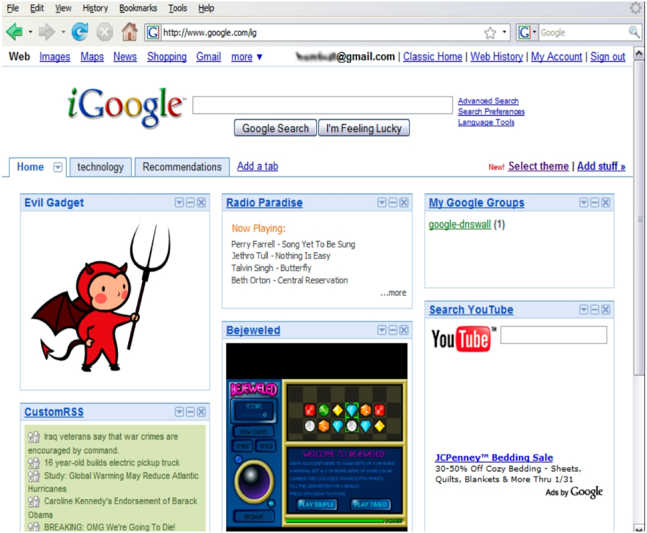
Mashup: *Applications obtained by mixing content from multiple providers*

- Individual contents being mixed - *Components*.
- Publisher of the mashup- *Host*.
- Execution environment- Web Browser.
- Web page (DOM) - Shared resource.
- Example: iGoogle, Facebook, Yelp

Example: iGoogle



Security Issue?

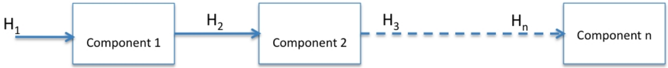


This study: Basic Mashups

Mashup with non-interacting components.

- Language: JavaScript (or any sequential imperative language).
  - Small-step Operational Semantics.
- Components: Programs $t_1; \ldots ; t_n$ in JavaScript.
- Mashup: Sequential composition - $t_1; \ldots ; t_n$.
- Shared Resource: Program heap.

Mashup Isolation Problem



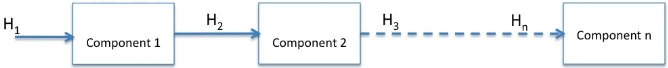
Verify/Enforce the following:

1. Host Isolation: No component must access any security-critical resources of the hosting page. Eg: window.location.
2. Inter-component Isolation: For all i, j, component i and j must access disjoint set of heap resources.

Our Previous Research (CSF'09, ESORICS'09):

- Enforces host isolation.
- Inter-component isolation is tricky:
  - Library functions are implicitly shared by components.
  - Need complete privilege separation.

Mashup Isolation Problem



Verify/Enforce the following:

1. Host Isolation: No component must access any security-critical resources of the hosting page. Eg: window.location.
2. Inter-component Isolation: For all i, j, component i and j must access disjoint set of heap resources.

Our Previous Research (CSF'09, ESORICS'09):

- Enforces host isolation.
- Inter-component isolation is tricky:
  - Library functions are implicitly shared by components.
  - Need complete privilege separation.

Capability Safe Languages

- **Main Idea**: *Every program carries certain capabilities which are the sole means for designating and accessing resources.*
- Object Capability languages (Mark Miller et al):
  - Capabilities idea applied to Object-oriented languages.
  - Properties: Connectivity begets Connectivtiy, No Authority Amplification, Defensive Consistency.
- Intuitively sounds very relevant, but we need formal definitions for carrying out rigorous proofs.

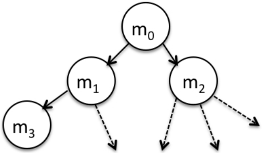
Plan

- Formally define Capability Systems for Prog. languages:
- Formally define Capability Safety.
- Derive a sufficient check for Inter-component isolation using Capability safety.

Capability Systems: Basic Features

Resources $(m_0, m_1, \ldots)$

- *Smallest granularity of readable/writable locations on the program heap.*
- Typically organized as a graph.



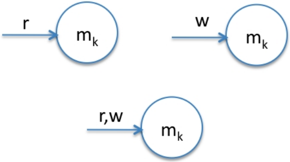
Subjects:

- *Entities that access resources.*
- Program expressions $t_0, t_1, \ldots$

Capability

Capability ($\mathcal{C}$)

- *Unforgeable entity that designates and provides access to a resource.*
- Pair (m, p) of resource m and permission $p \subseteq \{r, w\}$.



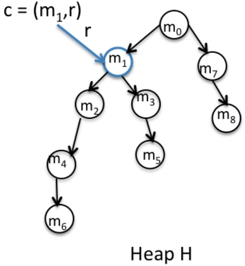
Subject-Capability Map $tCap$

- Each subject possesses certain capabilities.
- $tCap(t)$ is the set of capabilities associated with subject t.

Authority

Authority of a Capability (*cAuth*)

- *Upper-bound on resources that can be accessed using the capability.*
- $cAuth(H, c)$ is the authority of capability c w.r.t heap H.



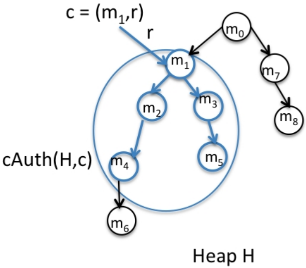
Authority of a Subject (*Auth*)

- Subjects possess capabilities which in turn provide authority.
- $Auth(H, t) = \bigcup_{c \in tCap(t)} cAuth(H, t)$ is the authority of subject t w.r.t heap H

Authority

Authority of a Capability (*cAuth*)

- *Upper-bound on resources that can be accessed using the capability.*
- $cAuth(H, c)$ is the authority of capability c w.r.t heap H.



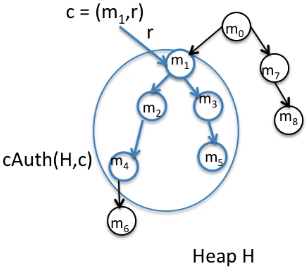
Authority of a Subject (*Auth*)

- Subjects possess capabilities which in turn provide authority.
- $Auth(H, t) = \bigcup_{c \in tCap(t)} cAuth(H, t)$ is the authority of subject t w.r.t heap H

Authority

Authority of a Capability (*cAuth*)

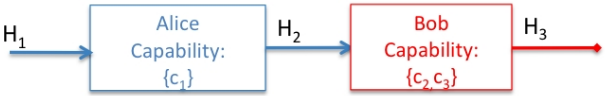
- *Upper-bound on resources that can be accessed using the capability.*
- $cAuth(H, c)$ is the authority of capability c w.r.t heap H.



Authority of a Subject (*Auth*)

- Subjects possess capabilities which in turn provide authority.
- $Auth(H, t) = \bigcup_{c \in tCap(t)} cAuth(H, t)$ is the authority of subject t w.r.t heap H

Achieving Mashup Isolation using Capabilities



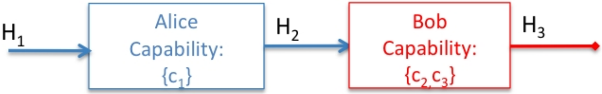
Idea: Inter-component isolation can be achieved by allocating capabilities with disjoint authority to Alice and Bob.

- Authority of a capability depends on the heap.
- Authorities must be disjoint with respect to what heap ?
  - $Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$ has to be checked
  - But we don't know H_2, we need a check on H_1 !

Next few slides

We define capablity safety and show that for safe systems, checking $Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$ is sufficient.

Achieving Mashup Isolation using Capabilities



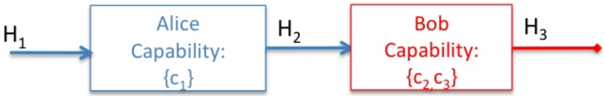
Idea: Inter-component isolation can be achieved by allocating capabilities with disjoint authority to Alice and Bob.

- Authority of a capability depends on the heap.
- Authorities must be disjoint with respect to what heap ?
  - $Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$ has to be checked
  - But we don't know H_2, we need a check on H_1 !

Next few slides

We define capablity safety and show that for safe systems, checking $Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$ is sufficient.

Achieving Mashup Isolation using Capabilities



Idea: Inter-component isolation can be achieved by allocating capabilities with disjoint authority to Alice and Bob.

- Authority of a capability depends on the heap.
- Authorities must be disjoint with respect to what heap ?
  - $Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$ has to be checked
  - But we don't know H_2, we need a check on H_1 !

Next few slides

We define capablity safety and show that for safe systems, checking $Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$ is sufficient.

Capability Safety

A capability system
[Capabilities, SubjectCapability Map, CapabilityAuthority Map]
is safe iff

1. All Access derives from Capabilities
2. Authority of a capability satisfies topology-only bounds
3. Only Connectivity begets Connectivity
4. No Authority Amplification

Other work considers a few more properties, our work focusses on the above 4 as they are sufficient for isolation.

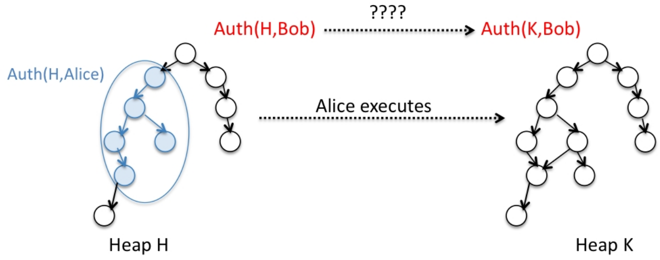
Capability Safety

A capability system
[Capabilities, SubjectCapability Map, CapabilityAuthority Map]
is safe iff

1. All Access derives from Capabilities
2. Authority of a capability satisfies topology-only bounds
3. Only Connectivity begets Connectivity
4. No Authority Amplification

Other work considers a few more properties, our work focusses on the above 4 as they are sufficient for isolation.

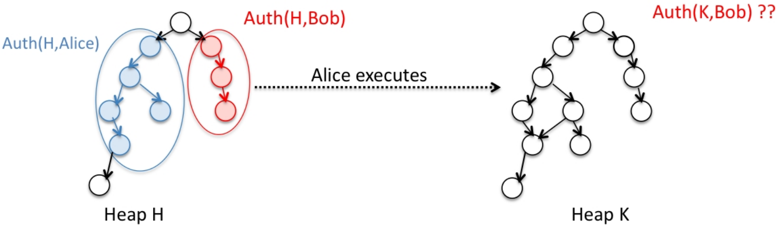
Authority Dynamics



Consider principals Alice and Bob.

- Alice executes and changes the heap from H to K.
- "Only Connectivity begets Connectivity" and "No Authority Amplification" give us a relation between $Auth(H, Bob)$ and $Auth(K, Bob)$.

Only Connectivity begets connectivity

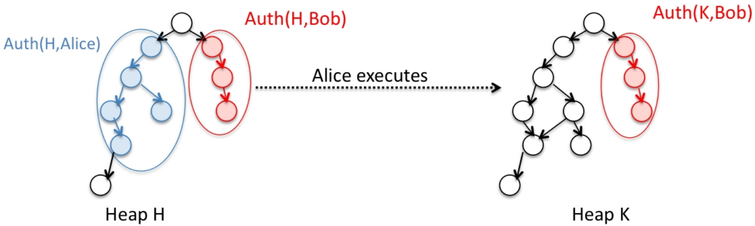


IF Bob's and Alice's authority with respect to H do not overlap

THEN Bob's authority stays the same

Formally, $Auth(K, Bob) = Auth(H, Bob)$

Only Connectivity begets connectivity

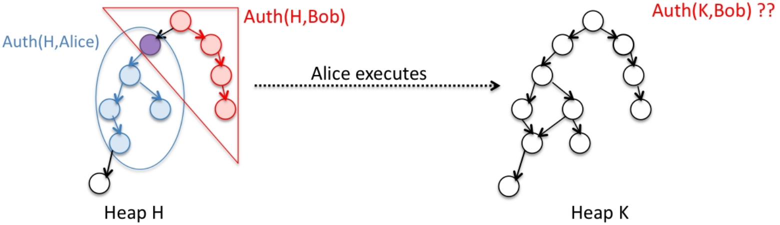


IF Bob's and Alice's authority with respect to H do not overlap

THEN Bob's authority stays the same

Formally, $Auth(K, Bob) \; = \; Auth(H, Bob)$

No Authority Amplification



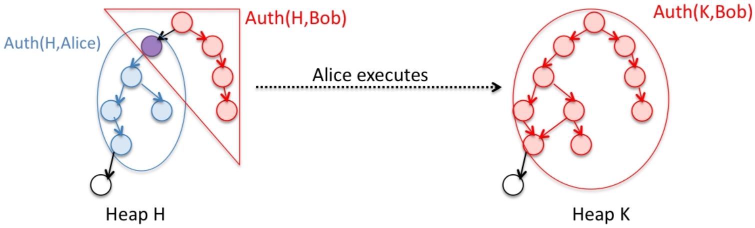
IF Bob's and Alice's authority with respect to H do overlap
THEN Bob's authority w.r.t K is at-most

- Both Alice's and Bob's authority w.r.t H.
- Any new authority created by Alice.

Formally,

$$Auth(K, Bob) \subseteq Auth(H, Bob) \bigcup Auth(H, Alice) \bigcup Act(K) \setminus Act(H)$$

No Authority Amplification



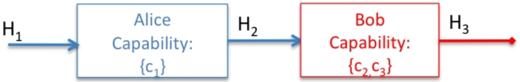
IF Bob's and Alice's authority with respect to H do overlap
THEN Bob's authority w.r.t K is at-most

- Both Alice's and Bob's authority w.r.t H.
- Any new authority created by Alice.

Formally,

$$Auth(K, Bob) \subseteq Auth(H, Bob) \bigcup Auth(H, Alice) \bigcup Act(K) \setminus Act(H)$$

Checking Inter-component Isolation



We want to prove $Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$

Initially,

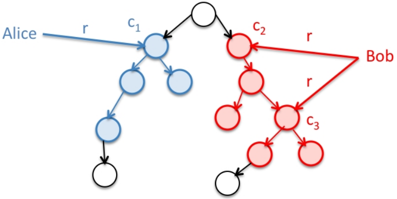
$Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$

$\Downarrow$

$Auth(H_2, Bob) = Auth(H_1, Bob)$

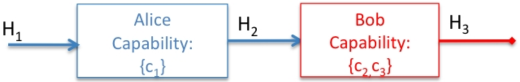
$\Downarrow$

$Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$



Heap H_1

Checking Inter-component Isolation



We want to prove $Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$

Initially,

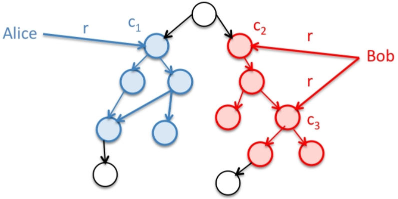
$$Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$$

$$\Downarrow$$

$$Auth(H_2, Bob) = Auth(H_1, Bob)$$

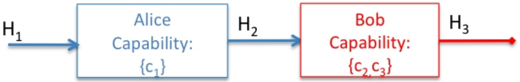
$$\Downarrow$$

$$Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$$



Heap H_2

Checking Inter-component Isolation



We want to prove $Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$

Initially,

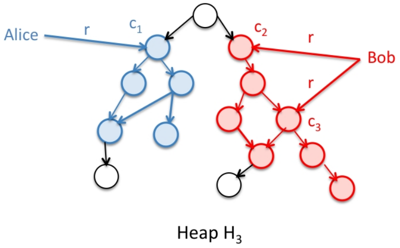
$$Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$$

$$\Downarrow$$

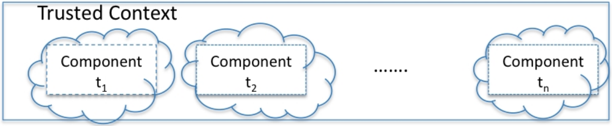
$$Auth(H_2, Bob) = Auth(H_1, Bob)$$

$$\Downarrow$$

$$Auth(H_1, Alice) \cap Auth(H_2, Bob) = \emptyset$$



Isolation Theorem



Definition: Authority-Isolation

For an initial heap H and components $t_1, \ldots, t_n$, authority isolation holds iff for all $i, j, i \neq j$:

$$Auth(H, t_i) \text{ and } Auth(H, t_j) \text{ do not overlap}$$

Theorem

Authority-Isolation $\Rightarrow$ Inter-component Isolation

Rigorously proven for any sequential imperative language, given its operational semantics.

Generalization: Authority Safety

Proof of Isolation theorem only requires a notion of authority of a subject- $Auth(H, t)$ such that

1. All resources accessed during the reduction of H, t are in $Auth(H, t)$.
2. *Auth* satisfies "Only Connectivity begets Connectivity".
3. *Auth* satisfies "No Authority Amplification".

We call the above 3 properties as Authority Safety.

- Capability systems provide a definition of authority $Auth(H, t) = \bigcup_{c \in tCap(t)} cAuth(H, t)$ but there could be other ways of defining authority.

Generalization: Authority Safety

Proof of Isolation theorem only requires a notion of authority of a subject- $Auth(H, t)$ such that

1. All resources accessed during the reduction of H, t are in $Auth(H, t)$.
2. *Auth* satisfies "Only Connectivity begets Connectivity".
3. *Auth* satisfies "No Authority Amplification".

We call the above 3 properties as Authority Safety.

- Capability systems provide a definition of authority $Auth(H, t) = \bigcup_{c \in tCap(t)} cAuth(H, t)$ but there could be other ways of defining authority.

Using the Isolation theorem in practice

Procedure for building safe Mashups

1. Prove that the underlying language is Capability Safe or Authority Safe.
2. Derive an enforcement function that provides Authority Isolation for different components.

Application: JavaScript Mashups

- Found a sub-language J_{safe} of JavaScript and proved Authority Safety for it.
- Derived an enforcement function that guarantees authority isolation.

Application: Google Caja Framework

- Formalized the core of Cajita $\subseteq$ JavaScript.
- Proved Capability Safety for the language Cajita.

Using the Isolation theorem in practice

Procedure for building safe Mashups

1. Prove that the underlying language is Capability Safe or Authority Safe.
2. Derive an enforcement function that provides Authority Isolation for different components.

Application: JavaScript Mashups

- Found a sub-language J_{safe} of JavaScript and proved Authority Safety for it.
- Derived an enforcement function that guarantees authority isolation.

Application: Google Caja Framework

- Formalized the core of Cajita $\subseteq$ JavaScript.
- Proved Capability Safety for the language Cajita.

Using the Isolation theorem in practice

Procedure for building safe Mashups

1. Prove that the underlying language is Capability Safe or Authority Safe.
2. Derive an enforcement function that provides Authority Isolation for different components.

Application: JavaScript Mashups

- Found a sub-language J_{safe} of JavaScript and proved Authority Safety for it.
- Derived an enforcement function that guarantees authority isolation.

Application: Google Caja Framework

- Formalized the core of Cajita $\subseteq$ JavaScript.
- Proved Capability Safety for the language Cajita.

J_{safe}: Enforcing Host Isolation

We want a subset of *JavaScript* which has a

1. Meaningful safe authority map
2. Supports an enforcement technique for enforcing authority isolation.

We start with subset J_{sub} defined in ESORICS'09.

- Subset defined using Filtering, Rewriting, Wrapping for preventing access of security-critical resources.
  - Filter eval, Rewrite e1[e2] to e1[IDX(e2)].
  - Wrap native functions ...
- Ensures that authority of any term does not contain security-critical resources.

J_{safe}: Enforcing Host Isolation

We want a subset of *JavaScript* which has a

1. Meaningful safe authority map
2. Supports an enforcement technique for enforcing authority isolation.

We start with subset J_{sub} defined in ESORICS'09.

- Subset defined using Filtering, Rewriting, Wrapping for preventing access of security-critical resources.
  - Filter eval, Rewrite e1[e2] to e1[IDX(e2)].
  - Wrap native functions ...
- Ensures that authority of any term does not contain security-critical resources.

J_{safe}: Enforcing Authority Isolation

Name space separation: Rename variables in different components into disjoint namespaces.

- Almost Works, but some authority overlap still exists.
  - Communication via naive objects.
    Alice: Alice_o.toString.channel = <msg>
    Bob: Bob_o.toString.channel
  - Communication using side-effect cause native functions.
    Alice: Alice_push = [].push; Alice_push(<msg>)
    Bob: Bob_pop = [].pop; Bob_pop()
- **Fix**:
  - Make native function objects readonly
  - Wrap native functions so that they never get the global object as the this object.

The resulting subset is called J_{safe}

J_{safe}: Enforcing Authority Isolation

Name space separation: Rename variables in different components into disjoint namespaces.

- Almost Works, but some authority overlap still exists.
  - Communication via naive objects.
    Alice: Alice_o.toString.channel = <msg>
    Bob: Bob_o.toString.channel
  - Communication using side-effect cause native functions.
    Alice: Alice_push = [].push; Alice_push(<msg>)
    Bob: Bob_pop = [].pop; Bob_pop()
- **Fix**:
  - Make native function objects readonly
  - Wrap native functions so that they never get the global object as the this object.

The resulting subset is called J_{safe}.

J_{safe}: Enforcing Authority Isolation

Name space separation: Rename variables in different components into disjoint namespaces.

- Almost Works, but some authority overlap still exists.
  - Communication via naive objects.
    Alice: Alice_o.toString.channel = <msg>
    Bob: Bob_o.toString.channel
  - Communication using side-effect cause native functions.
    Alice: Alice_push = [].push; Alice_push(<msg>)
    Bob: Bob_pop = [].pop; Bob_pop()
- **Fix**:
  - Make native function objects readonly
  - Wrap native functions so that they never get the global object as the this object.

The resulting subset is called J_{safe}.

J_{safe}: Enforcing Authority Isolation

Name space separation: Rename variables in different components into disjoint namespaces.

- Almost Works, but some authority overlap still exists.
  - Communication via naive objects.
    Alice: Alice_o.toString.channel = <msg>
    Bob: Bob_o.toString.channel
  - Communication using side-effect cause native functions.
    Alice: Alice_push = [].push; Alice_push(<msg>)
    Bob: Bob_pop = [].pop; Bob_pop()
- **Fix**:
  - Make native function objects readonly
  - Wrap native functions so that they never get the global object as the this object.

The resulting subset is called J_{safe}.

J_{safe} is authority safe

Main Contributions:

- We define an authority map $Auth_{J_{safe}}(H, t)$ for all heaps H and programs t.
- Theorem 1: $Auth_{J_{safe}}(H, t)$ is a safe authority map.
- Theorem 2: Namespace separation ensures authority isolation for J_{safe} programs.

Remarks:

- J_{safe} is more expressive than Facebook *FBJS* and Yahoo! *ADsafe*.
- Thinking in terms of authority helped us find new attacks on *FBJS* and *ADsafe*.
  - See Paper !

J_{safe} is authority safe

Main Contributions:

- We define an authority map $Auth_{J_{safe}}(H, t)$ for all heaps H and programs t.
- Theorem 1: $Auth_{J_{safe}}(H, t)$ is a safe authority map.
- Theorem 2: Namespace separation ensures authority isolation for J_{safe} programs.

Remarks:

- J_{safe} is more expressive than Facebook *FBJS* and Yahoo! *ADsafe*.
- Thinking in terms of authority helped us find new attacks on *FBJS* and *ADsafe*.
  - See Paper !

Results and Future Work

Results:

- Capability Safety $\Rightarrow$ Authority Safety $\Rightarrow$ Isolation.
- J_{safe} is Authority safe.
- Cajita is Capability safe.

Future Work:

- Define the isolation problem for mashups with interacting components.
- Formalize other aspects of capability systems:
  - absolute encapsulation, defensive consistency
  - Use the above for controlling interaction between components.
- New proof technique for authority isolation (Separation Logic)