

Object Capabilities and Isolation of Untrusted Web Applications

Ankur Taly

Dept. of Computer Science, Stanford University

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

Outline

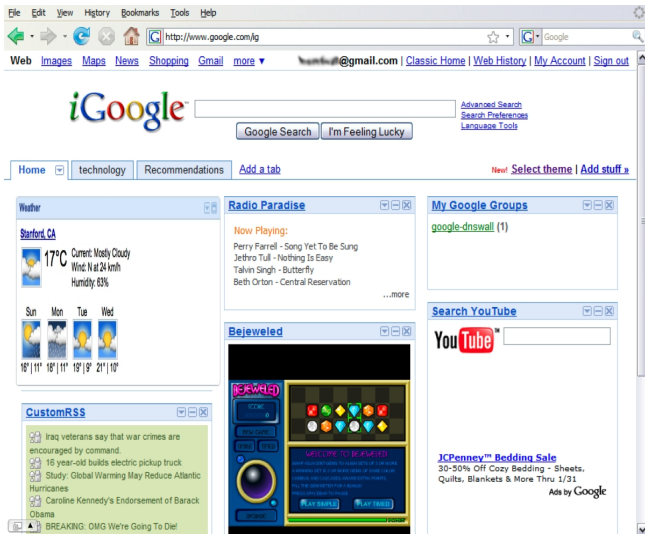
- 1 Isolation problem for Web Mashups
- 2 Formal definition of Capability Safe languages
- 3 Solving the Isolation problem using Capability Safe languages
- 4 Application: JavaScript Mashups

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



File Edit View History Bookmarks Tools Help

http://www.google.com/ig

Web Images Maps News Shopping Gmail more ▾ [kankat@gmail.com](#) | [Classic Home](#) | [Web History](#) | [My Account](#) | [Sign out](#)


iGoogle [Advanced Search](#) [Search Preferences](#) [Language Tools](#)





Google Search I'm Feeling Lucky

Home ▾ technology Recommendations [Add a tab](#) New! [Select theme](#) | [Add stuff](#) ▾

Weather

[Stanford, CA](#)

 **17°C** Current: Mostly Cloudy
Wind: N at 24 km/h
Humidity: 63%

Sun	Mon	Tue	Wed
			
16° 11°	18° 11°	19° 9°	21° 10°

Radio Paradise

Now Playing:

Perry Farrell - Song Yet To Be Sung
Jethro Tull - Nothing Is Easy
Talvin Singh - Butterfly
Beth Orton - Central Reservation
...more

My Google Groups


[google-dnswall \(1\)](#)

Search YouTube





YouTube

JCPenney™ Bedding Sale
30-50% Off Cozy Bedding - Sheets, Quilts, Blankets & More Thru 1/31
Ads by Google

Bejeweled



CustomRSS

-  Iraq veterans say that war crimes are encouraged by command.
-  16 year-old bulks electric pickup truck
-  Study: Global Warming May Reduce Atlantic Hurricanes
-  Caroline Kennedy's Endorsement of Barack Obama

BREAKING: OMG We're Going To Die!

Security Issue?

The screenshot shows a web browser window displaying the Google iGoogle homepage. The browser's address bar shows the URL `http://www.google.com/ig`. The page features the iGoogle logo, a search bar, and navigation links such as "Advanced Search", "Search Preferences", and "Language Tools". Below the search area, there are tabs for "Home", "technology", "Recommendations", and "Add a tab".

The main content area is populated with several web mashups:

- Evil Gadget:** A cartoon illustration of a red devil character with horns, wings, and a tail, holding a pitchfork.
- Radio Paradise:** A music player interface showing "Now Playing:" with a list of songs including "Song Yet To Be Sung" by Perry Farrell, "Nothing Is Easy" by Jethro Tull, "Butterfly" by Talvin Singh, and "Central Reservation" by Beth Orton.
- My Google Groups:** A section titled "google-dnswall (1)".
- Search YouTube:** A search interface for YouTube with the YouTube logo and a search input field.
- Bejeweled:** A game interface for Bejeweled, showing a grid of colored gems and a "WELCOME TO BEJEWELD" message.
- CustomRSS:** A feed of RSS items with headlines such as "Iraq veterans say that war crimes are encouraged by command.", "16 year-old builds electric pickup truck", "Study: Global Warming May Reduce Atlantic Hurricanes", "Caroline Kennedy's Endorsement of Barack Obama", and "BREAKING: OMG We're Going To Die!".

At the bottom right of the page, there is an advertisement for "JCPenney™ Bedding Sale" with the text "30-50% Off Cozy Bedding - Sheets, Quilts, Blankets & More Thru 1/31" and "Ads by Google".

This study: Basic Mashups

Mashup with non-interacting components.



- Language: JavaScript (or any sequential imperative language).
 - Small-step Operational Semantics.
- Components: Programs $t_1; \dots; t_n$ in JavaScript.
- Mashup: **Sequential composition** - $t_1; \dots; t_n$.
- 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 Connectivity, No Authority Amplification, Defensive Consistency.
- Intuitively sounds very relevant, but we need formal definitions for carrying out rigorous proofs.

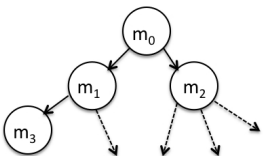
Plan

- 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, \dots)

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



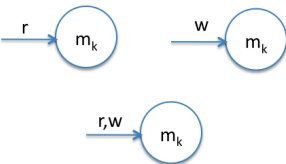
Subjects:

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

Capability

Capability (\mathcal{C})

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



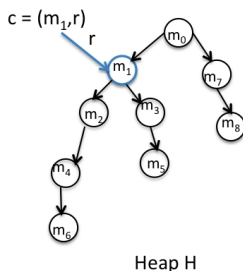
Subject-Capability Map $tCap$

- 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 .



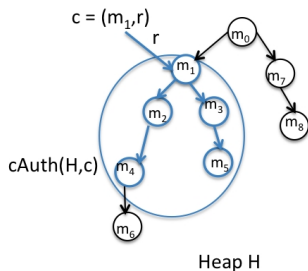
Authority of a Subject ($Auth$)

- 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 .



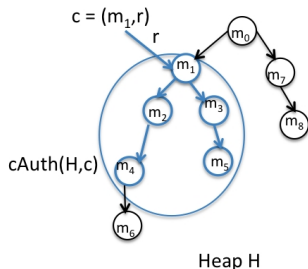
Authority of a Subject ($Auth$)

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

Authority

Authority of a Capability ($cAuth$)

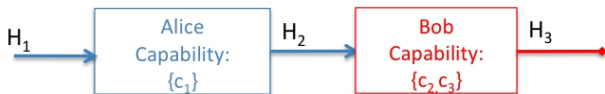
- Upper-bound on resources that can be accessed using the capability.
- $cAuth(H, c)$ is the authority of capability c w.r.t heap H .



Authority of a Subject ($Auth$)

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

Achieving Mashup Isolation using Capabilities



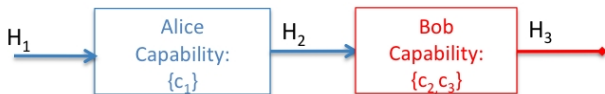
Idea: Inter-component isolation can be achieved by allocating capabilities with **disjoint authority** to Alice and Bob.

- 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 **capability 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



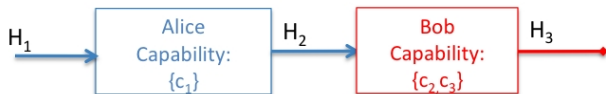
Idea: Inter-component isolation can be achieved by allocating capabilities with **disjoint authority** to Alice and Bob.

- 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 **capability 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 **capability 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

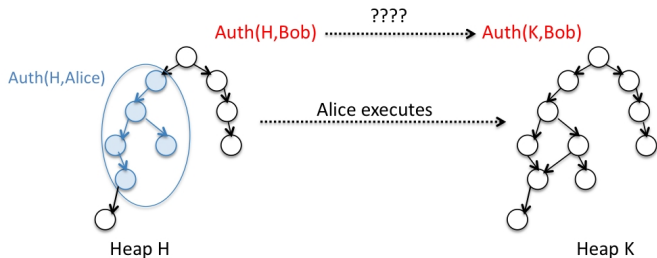
[Capabilities, SubjectCapability Map, CapabilityAuthority Map]

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.

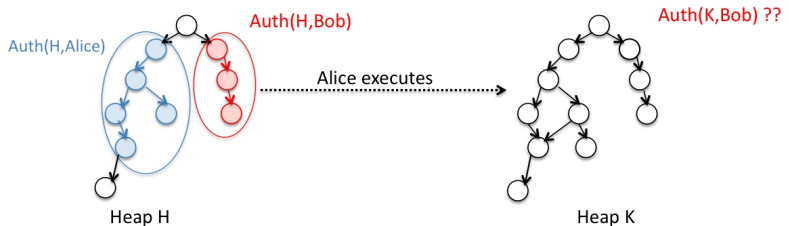
Authority Dynamics



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

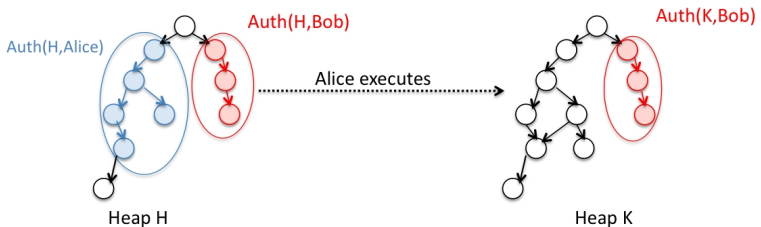


IF Bob's and Alice's authority with respect to H do not overlap

THEN Bob's authority stays the same

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

Only Connectivity begets connectivity

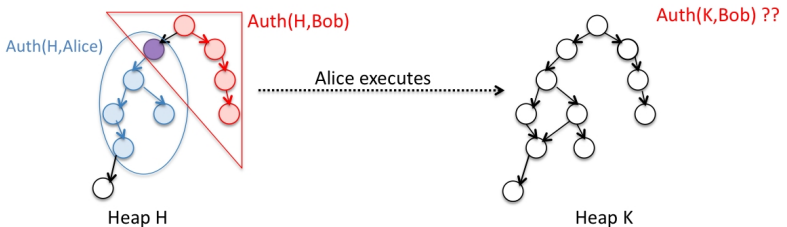


IF Bob's and Alice's authority with respect to H do not overlap

THEN Bob's authority stays the same

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

No Authority Amplification



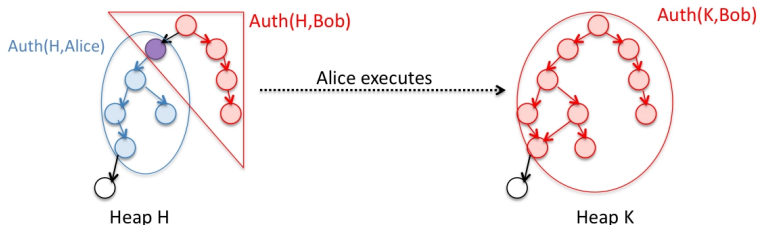
IF Bob's and Alice's authority with respect to H do overlap
 THEN Bob's authority w.r.t K is at-most

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

Formally,

$$\text{Auth}(K, \text{Bob}) \subseteq \text{Auth}(H, \text{Bob}) \cup \text{Auth}(H, \text{Alice}) \cup \text{Act}(K) \setminus \text{Act}(H)$$

No Authority Amplification



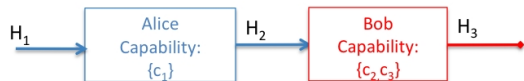
IF Bob's and Alice's authority with respect to H do overlap
 THEN Bob's authority w.r.t K is **at-most**

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

Formally,

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

Checking Inter-component Isolation



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

Initially,

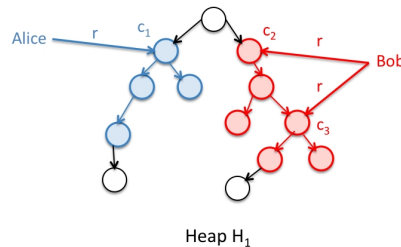
$$Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$$



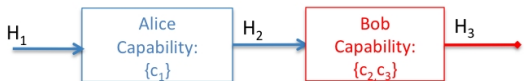
$$Auth(H_2, Bob) = Auth(H_1, Bob)$$



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



Checking Inter-component Isolation



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

Initially,

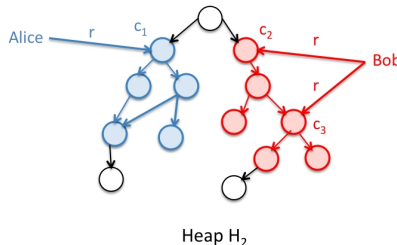
$$Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$$

\Downarrow

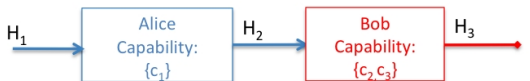
$$Auth(H_2, Bob) = Auth(H_1, Bob)$$

\Downarrow

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



Checking Inter-component Isolation



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

Initially,

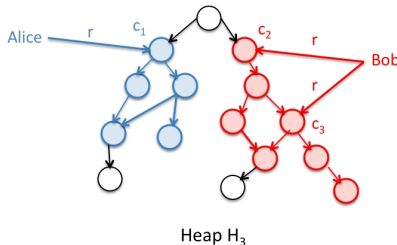
$$Auth(H_1, Alice) \cap Auth(H_1, Bob) = \emptyset$$

\Downarrow

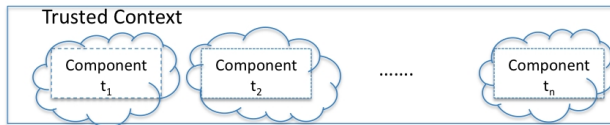
$$Auth(H_2, Bob) = Auth(H_1, Bob)$$

\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, \dots, t_n , authority isolation holds iff for all $i, j, i \neq j$:

$Auth(H, t_i)$ and $Auth(H, t_j)$ 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.

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)